2 Function Reference

Overview

The Function Reference section provides technical descriptions of functions available in DSPower series and Hypersignal-Macro series software. Functions are further subdivided into the following categories:

- DSP Engine Functions
- Library Functions
- DSP Source Code Functions

Note that DSP Engine functions are not the same as block functions. Blocks, which begin as template files and become executable DSP Engine programs in the DSPower terminology, are covered separately in chapter 1, Block Reference.

See Also

Function Alphabetic List and Cross Reference

Below is an alphabetic listing of all functions, cross-referenced by page number.

Function Categories

DSP Engine Functions

Library Functions

DSP Source Code Functions

Function Alphabetic List and Cross Reference

The table below provides a cross-reference for all DSPower series library functions to their corresponding Reference Guide descriptions, which contain parameter information and calling instructions, comments, related functions, and examples. Each Reference Guide description also contains a "Related Users Guide Topic" section which lists related areas in the Signalogic DSP Software Users Guide. Users Guide information typically is more general in nature, including more extensive source code examples and explanation of relationship with other functions and software components.

Function Name	Page
DSAcquireWvfrmFile	198
DSAddEngMgrStatusLine	139
DSAssignBoard	198
DSCalcSampFreq	201
DSCancelWaitBuffer	204
DSCancelWaitFlag	204
DSDisableBoard	207
DSDSPToIEEE	210
DSEngineOpen	140
DSEngineClose	142
DSExecEngineFunction	144
DSFreeBoard	209
DSGenerateWvfrmFile	213
DSGetBoardBaseAddr	218

DSGetBoardBusType 214
DSGetBoardClass 216
DSGetBoardInfo 216
DSGetEngineErrorStatus 147
DSGetEnginePath 149
DSGetEngineUsage 151
DSGetEngineState 151
DSGetEngMgrErrorStatus 153
DSGetHVarMem 219
DSGetMem 221
DSGetMemArch 224
DSGetMemSize 229
DSGetHWMgrCurVal 168
DSGetHWMgrEntry 169
DSGetHWMgrEntryIndex 171
DSGetHWMgrErrorStatus 173
DSGetHWMgrNumEntries 175
DSGetHWMgrVar 177
DSGetHWSetupFile 179
DSGetWordLength 232
DSHideEngMgrStatusWindow 156
DSHoldBoard 230
DSHoldProcessor 234
DSInitBoard 235
DSIEEEToDSP 239
DSLoadFileBoard 237
DSLoadFileProcessor 241
DSPutHVarMem 243
DSPutMem 246
DSReadBoardReg 254
DSReadHWSetupFile 179

DSRegisterEngineMsgWnd 157
DSResetBoard 254
DSResetProcessor 255
DSRunBoard 257
DSRunProcessor 258
DSSendEngineCommand 159
DSSetBoardBaseAddr 260
DSSetBoardBusType 260
DSSetEngineTemplatePath 161
DSSetEngineWaveformPath 163
DSSetHWMgrEntry 185
DSSetHWMgrCurVal 183
DSSetHWMgrVar 188
DSSetHWSetupFile 190
DSShowEngMgrStatusWindow 164
DSShowHardwareSelectorDlg 190
DSUnregisterEngineMsgWnd 165
DSWaitForBuffer 263
DSWaitForFlag 265
DSWriteBoardReg 270
DSWriteHWSetupFile 190

2.1 DSP Engine Functions

DSP Engine simulation functions have been further subdivided into the following categories:

- Hypersignal-Macro series software functions
- MATLAB functions

Within each category, functions are listed alphabetically. Note that with the online Help version of documentation, you can use the Browse buttons (">>" and "<<") to move forward and backwards between different function descriptions.

For a complete listing of all DSPower series software functions, see Function Alphabetic List and Cross Reference, above.

NOTE: In function descriptions listed below, knowledge of Hypersignal macro language and MATLAB .m program definition, variable declarations, and syntax is assumed.

DSP Engine Function Categories

2.2 Library Functions

Library functions have been further subdivided into the following categories:

- Engine Manager Functions (enmgr.dll, enmgr32.dll)
- Hardware Manager Functions (hwmgr.dll, hwmgr32.dll)
- Hardware Library Functions (hwlib.dll, hwlib32.dll)
- Signal Processing Library Functions (splib.dll)

Within each category, functions are listed alphabetically. Note that with the online Help version of documentation, you can use the Browse buttons (">>" and "<<") to move forward and backwards between different function descriptions.

For a complete listing of all DSPower series software functions, see Function Alphabetic List and Cross Reference, above.

NOTE: In the function descriptions listed below, knowledge of type definitions declared in the standard Microsoft windows.h include file is assumed. For example, LPCSTR is defined as const char far*, UINT is defined as unsigned int, etc.

Library Function Categories

Related Users Guide Chapters

Using Engine Manager

Using Hardware Manager

Using DSPower-HWLib and DSPower-SPLib Software

2.2.1 Engine Manager Functions

[@k functions, Engine Manager, enmgr.dll, enmgr32.dll]

Below are listed functions in the Engine Manager API. For information on how to include Engine Manager (enmgr.dll or enmgr32.dll) in C/C++, Visual Basic, MATLAB, and LabView projects, see Signalogic DSP Software Users Guide section 7.1, Engine Manager Access.

Engine Manager Functions

DSAddEngMgrStatusLine

DSEngineOpen

DSEngineClose

DSExecEngineFunction

DSGetEngineErrorStatus

DSGetEnginePath

DSGetEngineUsage

DSGetEngineState

DSGetEngMgrErrorStatus

DSHideEngMgrStatusWindow

DSRegisterEngineMsgWnd

DSSendEngineCommand

DSSetEngineTemplatePath

DSSetEngineWaveformPath

DSShowEngMgrStatusWindow
DSUnregisterEngineMsgWnd

DSAddEngMgrStatusLine

HWND DSAddEngMgrStatusLine (pszText)

LPCSTR pszText; // pointer to text string to add to error/status window

The DSAddEngMgrStatusLine function adds a line of text to the Engine Manager error/status listbox. Text is always added to the end of any text currently in the listbox, regardless of whether the error/status listbox is currently enabled or not.

Parameter	Description
pszText	pointer to null-terminated text string to add to Engine Manager error/status listbox

Returns

The return value is the handle of the error/status listbox window, if successful. Otherwise, the return value is NULL, indicating that the error/status box could not be created or updated with new information.

Comments

The error/status listbox is created if no text has previously been added or displayed.

Text is added to the Engine Manager error/status listbox regardless of whether the listbox is visible or not. Text is always added to the end of the current listbox contents, which are scrolled up, if necessary, to ensure the most recently added text is in view.

C/C++ Example

```
hwnd = DSAddEngMgrStatusLine("processing buffer-ready message");
```

```
if (hwnd != NULL) {

    SetMenu(hwnd, hMenu);  // add application-specific menu to status box
}
```

See Also

DSHideEngMgrStatusWindow

DSShowEngMgrStatusWindow

Related Users Guide Topics

section 7.5.1 Internal Error Handling in Engine Manager

section 7.5.2 Engine Error Handling

section 7.5.3 Error/Status Window in Engine Manager

DSEngineOpen

HENGINE DSEngineOpen (pszEnginePath, pszWaveformPath)

```
LPCSTR  pszEnginePath;      // path where engine is located
LPCSTR  pszWaveformPath;    // path of engine waveform files
```

The DSEngineOpen function opens and initializes the specified DSP Engine, and returns an engine handle which can be used in numerous other Engine Manager functions, as well as some Hardware Library and Signal Processing Library functions. Some pre-defined constants can be specified in place of **szEnginePath**, as shown below:

Parameter	Description
pszEnginePath	pointer to null-terminated string containing partially- or fully-qualified path of DSP Engine to open; alternatively, the constants defined below can be used:

Value	Meaning
DS_EO_HSM	search for and open Hypersignal-Macro as DSP Engine
DS_EO_HSA	search for and open Hypersignal-

		Acoustic as DSP Engine
	DS_EO_HSMNRT	search for and open Hypersignal-Macro NRT as DSP Engine
	DS_EO_MAT	search for and open MATLAB as DSP Engine
pszWaveformPath	pointer to null-terminated string containing fully-qualified path location of waveform files which might be accessed by the DSP Engine	

Returns

The return value is non-zero, representing a valid engine handle if successful. Otherwise, the return value is NULL, indicating that the engine could not be located (path was not valid), or the engine could not be initialized. For further error checking, use the DSGetEngMgrErrorStatus routine (see C/C++ source code example below).

Comments

A partially specified engine path will cause the DSEngineOpen function to assume an engine type and perform a rudimentary search. For example, specifying only "matlab" in the engine path will cause the DSEngineOpen function to attempt to open :x:\matlab\bin\matlab.exe", where "x" is the drive location of the Engine Manager library.

The waveform path is not checked for valid waveform files, other than to ensure that the path actually exists.

The DSEngineOpen call opens DSP Engines such as Hypersignal-Macro series or MATLAB programs and places them in command mode, ready to accept instructions. DSP Engines are opened so as not to be visible (and conserve memory by not loading graphics drivers or other user-interface objects, if possible). Internally, DSP Engines are opened and initialized using either a Windows WinExec call or functions inside shell.dll. If the WinExec call is used, the "show state" parameter is set to SW_HIDE, and in the case of DOS engines, a .pif file may be associated with the engine.

For more information, see also Signalogic DSP Software Users Guide section 7.2.1, Opening DSP Engines.

C/C++ Example

```
HENGINE hEngine;

hEngine = DSEngineOpen(szEnginePath, szWaveformPath);
if (!hEngine) {
      nErrorType = DSGetEngMgrErrorStatus();
      ...do error processing...
}
```

See Also

DSEngineClose

DSGetEngineUsage

DSSetEngineWaveformPath

DSSetEngineTemplatePath

DSExecEngineFunction

Related Users Guide Topics

section 7.2 Controlling DSP Engines

section 7.2.1 Opening DSP Engines

section 7.2.2 Sending DSP Engine Commands

section 7.3 Executing Functions in DSP Engines

DSEngineClose

UINT DSEngineClose (hEngine)

HENGINE hEngine; // engine handle

The DSEngineClose function closes the specified DSP Engine and invalidates the engine handle.

Parameter	Description
hEngine	handle to engine obtained from previous DSEngineOpen call

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid, or the engine could not be closed. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines (see C/C++ source code example below).

Comments

The Engine Manager library is designed to close any engines still open when it is unloaded from the system. Unloading occurs only when all applications which implicitly or explicitly loaded Engine Manager have terminated. However, for each application it is always a good idea to call DSEngineClose for any currently active engine handles just prior to terminating. This avoids unnecessary consumption of system resources should Engine Manager not be unloaded immediately because other applications are still using it.

Also it is good programming practice to set the hEngine parameter to NULL after a successful DSEngineClose, to avoid the possibility of accidentally using an engine handle value in a Hardware Library call which might have been reassigned to another engine, in use by another application.

For more information details about obtaining engine handles using the DSEngineOpen call, see also Signalogic DSP Software Users Guide section 7.2.1, Opening DSP Engines.

C/C++ Example

```
if (!DSEngineClose(hEngine)) {

    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetEngMgrErrorStatus();
    ...do error processing...
}
else hEngine = NULL;
```

See Also

DSEngineOpen

DSGetEngineUsage

DSSetEngineWaveformPath

DSSetEngineTemplatePath

DSExecEngineFunction

Related Users Guide Topics

section 7.2 Controlling DSP Engines

section 7.2.1 Opening DSP Engines

section 7.2.2 Sending DSP Engine Commands

section 7.3 Executing Functions in DSP Engines

DSExecEngineFunction

UINT DSExecEngineFunction (hEngine, pszCommandStr, uMode)

```
HENGINE  hEngine;          // engine handle
LPCSTR  pszCommandStr;     // command string of function to execute
UINT  uMode;               // mode of execution
```

The DSExecEngineFunction call sends a text string containing an function, along with any associated input parameters, to the specified engine for execution. The command string must be null-terminated and can be as long as needed.

Parameter	Description
hEngine	engine handle which was previously obtained via call to DSEngineOpen
pszCommandStr	pointer to null-terminated command string containing engine function to be executed, and any associated input parameters
uMode	mode in which the engine function should be executed; one of the possible pre-defined values listed below should be specified:

Value	Meaning
DS_EEF_ASYNC	execute engine function asynchronously with respect to calling application (calling application continues to execute

	simultaneously); this is the default used if uMode is left NULL
DS_EEF_SYNC	execute engine function synchronously with respect to calling application (calling application waits for engine function to complete)

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid, or the command could not be executed. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines (see C/C++ source code example below).

Comments

DSExecEngineFunction is a general purpose function which allows DSP Engine functions to be invoked from user-defined calling application programs. This facility is not to be confused with DSP Engine functions, or combinations of functions, which are executed automatically and transparently as a result of standard API calls inside the DSPower-HWLib or DSPower-SPLib libraries.

The purpose of the DSExecEngineFunction call is to allow engine functions to be called directly, assuming knowledge of the syntax and parameters of the engine call.

Either synchronous or asynchronous operation can be specified using the uMode parameter. Synchronous operation means the calling application waits for the engine function to complete; asynchronous operation means the calling application continues to execute simultaneously.

C/C++ Example 1

Below is an example which evaluates a difference equation using Hypersignal-Macro software as the engine. The difference equation creates the output waveform file "sp_avg" from the input waveform file "speech":

```
HENGINE hEngine;
int nErrorType;

hEngine = DSEngineOpen(DS_EO_HSM, "c:\\hsmacro\\waveform");
```

```
if (!hEngine) {
    nErrorType = DSGetEngMgrErrorStatus();
    ...do error processing...
}

DSExecEngineFunction(hEngine, "f_diffeq(\"sp_avg\", \"y[n] = y[n] + (x1[n] – x1[n-
M])/Fr\", \"speech\", \"\", 0, 0, 0)", DS_EEF_ASYNC);
```

In the above example, the difference equation was executed asynchronously, meaning that the calling application continued to run at the same time. In such a case, the calling application can register a window using the DSRegisterEngineMsgWnd function to receive a "program done" message when the function is complete.

Note the use of "\"" to specify quote marks within the command string for string parameters required by the f_diffeq engine function. For a list of DSPower series software "f_xxxx" engine functions, see section 2.1 above, DSP Engine Functions.

C/C++ Example 2

The example below uses a combination of a DSPower engine function and a built-in MATLAB function to execute a simple FFT:

```
HENGINE hEngine;
int nErrorType;

hEngine = DSEngineOpen(DS_EO_MAT, "c:\\matlab\\waveform");

if (!hEngine) {
    nErrorType = DSGetEngMgrErrorStatus();
    ...do error processing...
}

DSExecEngineFunction(hEngine, "x = f_load(\"speech.wav\")", DS_EEF_SYNC);
DSExecEngineFunction(hEngine, "y = fft(x)", DS_EEF_ASYNC);
```

The above example shows the DSPower series software f_load function being used to load the time domain waveform file "speech.wav". Note also this is done synchronously, meaning the calling application would wait for the engine to complete the f_load function. The second DSExecEngineFunction simply passes a command string invoking a MATLAB Signal Processing Toolbox function directly, allowing the calling application to continue executing by specifying the DS_EEF_ASYNC constant. In such a case, the calling application can register a window using the DSRegisterEngineMsgWnd function

to receive a "program done" message when the engine function is complete.

C/C++ Example 3

The example below shows the f_fft DSPower function being used to accomplish the same result as shown in Example 2 above, but in one step:

```
HENGINE hEngine;
int nErrorType;

hEngine = DSEngineOpen(DS_EO_MAT, "c:\\matlab\\waveform");

if (!hEngine) {
    nErrorType = DSGetEngMgrErrorStatus();
    ...do error processing...
}

DSExecEngineFunction(hEngine, "y = f_fft(\"speech.wav\")", DS_EEF_ASYNC);
```

See Also

DSRegisterEngineMsgWnd

DSSendEngineCommand

DSSetEngineTemplatePath

DSAcquireWvfrmFile

DSGenerateWvfrmFile

WM_DSPENGINE_PROGSTATUS

Related Users Guide Topics

section 7.3 Executing Functions in DSP Engines

section 7.4 Registering Callback Messages from DSP Engines

Program Status Messages from DSP Engines (inside section 7.4)

section 7.2.2 Sending DSP Engine Commands

DSGetEngineErrorStatus

UINT DSGetEngineErrorStatus **(hEngine)**

```
HENGINE  hEngine;                    // engine handle
```

The DSGetEngineErrorStatus function retrieves the current error status for the specified engine.

Parameter	Description
hEngine	handle to engine obtained from previous DSEngineOpen call

Returns

The return value indicates the current error condition for the specified engine, if one exists; otherwise, the return value is NULL. For DSPower-HWLib or DSPower-SPLib functions which require an engine handle and which return NULL, DSGetEngineErrorStatus should be checked first, followed by DSGetEngMgrErrorStatus (see C/C++ source code example below).

Comments

The Engine Manager (enmgr.dll or enmgr32.dll) error handling scheme is straightforward. Most calls return a non-zero value on success, and a zero (NULL) on failure. When calls into Engine Manager fail, there are two levels of error possibilities: the library itself (or "base level"), and engine-related. Engine-related error conditions are returned by the DSGetEngineErrorStatus call, while base-level error conditions are returned by DSGetEngMgrErrorStatus.

A current error condition is maintained for each active DSP Engine. If an error condition occurs but is not retrieved, and further engine-related calls are made which cause *another* error condition, DSGetEngineErrorStatus returns only the most recent error condition.

Engine Manager also maintains a "status listbox" which can be made visible using the DSShowEngMgrStatusWindow call. This status window shows a history of error conditions from DSPower series library calls.

For more information, see Signalogic DSP Software Users Guide sections 7.5.1, Internal Error Handling in Engine Manager, 7.5.2, Engine Error Handling, and 7.5.3, Error/Status Window in Engine Manager.

C/C++ Example

Below is an example showing error investigation in case of failure of a DSAssignBoard call, which is located inside Hardware Library:

```
int nErrorType;

if (!DSAssignBoard(hEngine, szBoardDesignator, uBusType, uIOBaseAddr,
uMemBaseAddr)) {

    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetEngMgrErrorStatus();
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
```

Note that the DSGetEngineErrorStatus function should only be used with a valid engine handle. Otherwise, it is possible to incur an "invalid engine handle" error, which may overwrite a previously existing error condition. See **Comments** above.

See Also

DSGetEngMgrErrorStatus

DSGetHWMgrErrorStatus

DSGetHWLibErrorStatus

DSEngineOpen

Related Users Guide Topics

section 7.5 Error Handling in Engine Manager

section 7.5.1 Internal Error Handling in Engine Manager

section 7.5.2 Engine Error Handling

section 7.5.3 Error/Status Window in Engine Manager

DSGetEnginePath

UINT DSGetEnginePath (hEngine,
pszEnginePath)

HENGINE hEngine; // engine handle

LPSTR pszEnginePath; // engine path to be filled in and returned

The DSGetEnginePath function retrieves the path for the specified DSP Engine and returns the value in pszEnginePath.

Parameter	Description
hEngine	handle to engine obtained from previous DSEngineOpen call
pszEnginePath	pointer to string variable which will be filled in with the specified engine path and returned as a null-terminated string

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines (see C/C++ source code example below).

Comments

The string variable pointed to by pszBoardDesignator is assumed to allow enough space for a return value (no space validation of any kind is performed). At least 144 characters is required (see C/C++ source code example below).

Essentially, the DSGetEnginePath function should return the same path used in the original DSEngineOpen call. The value might be different if a pre-defined constant or partial path was used in the DSEngineOpen call.

C/C++ Example

```
char szEnginePath[144];

if (!DSGetEnginePath(hEngine, szEnginePath)) {

    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetEngMgrErrorStatus();
    ...do error processing...
}
```

See Also

DSEngineOpen

DSSetEngineWaveformPath

DSSetEngineTemplatePath

Related Users Guide Topics

section 7.2 Controlling DSP Engines

section 7.2.1 Opening DSP Engines

DSGetEngineState

UINT DSGetEngineState (hEngine)

HENGINE hEngine; // engine handle

The DSGetEngineState function retrieves the current state of the specified DSP Engine.

Parameter	Description
hEngine	handle to engine obtained from previous DSEngineOpen call

Returns

The return value is NULL if the engine handle was invalid. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines (see C/C++ source code example below). Otherwise, the return value is as defined below:

Value	Meaning
DS_ES_RUNNING	engine is in running (busy) state
DS_ES_IDLE	engine is in idle state
DS_ES_STOPPENDING	engine is running, but a stop (shutdown) command is pending
DS_ES_STOPPED	engine is stopped (shutdown)

Comments

The DSGetEngineState call can be used to learn the current state for a specified DSP Engine. Note that state changes can be determined dynamically by using DSRegisterEngineMsgWnd to register a window to receive WM_DSPENGINE_ENGINESTATE messages.

C/C++ Example

```
char szText[50];
short int nErrorType;

switch (DSGetEngineState(hEngine)) {

    case DS_ES_RUNNING:

        _fstrcpy(szText, "Running");
        break;

    case DS_ES_IDLE:

        _fstrcpy(szText, "Idle");
        break;

    case DS_ES_STOPPENDING:

        _fstrcpy(szText, "Stop Pending");
        break;

    case DS_ES_STOPPED:

        _fstrcpy(szText, "Shutdown");
        break;

    default:

        _fstrcpy(szText, "Error in DSGetEngineState Call");
        nErrorType = DSGetEngineErrorStatus(hEngine);
        if (nErrorType == NULL) nErrorType = DSGetEngMgrErrorStatus();

        // do error processing...

        break;
}
MessageBox(NULL, szText, "Current Engine State", MB_OK);
```

See Also

WM_DSPENGINE_ENGINESTATE

DSRegisterEngineMsgWnd

DSEngineOpen

DSEngineClose

Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Status Messages from DSP Engines

section 7.2 Controlling DSP Engines

section 7.2.1 Opening DSP Engines

DSGetEngineUsage

UINT DSGetEngineUsage ( )

The DSGetEngineUsage function returns the number of currently opened DSP Engines by the calling application.

Parameter	Description
None	

Returns

The return value is zero if no engines are currently opened by the calling application. Otherwise, the return value is a positive value indicating the number of engines currently open.

Comments

After invoking a DSEngineClose function for each open engine, DSGetEngineUsage should return 0.

One possible use for this call is in case of catastrophic engine failure, when it may be important to learn if one or more engines has been forced to close due to internal Engine Manager error processing.

C/C++ Example

```
UINT uNumEngines;
char szText[30];

uNumEngines = DSGetEngineUsage();
```

```
itoa(uNumEngines, szText, 10);
MessageBox(NULL, szText, "Number of open engines", MB_OK);
```

See Also

DSEngineOpen

DSEngineClose

Related Users Guide Topics

section 7.2 Controlling DSP Engines

section 7.2.1 Opening DSP Engines

DSGetEngMgrErrorStatus

UINT DSGetEngMgrErrorStatus ( )

The DSGetEngMgrErrorStatus function retrieves the current base-level error status for the Engine Manager library(enmgr.dll or enmgr32.dll).

Parameter	Description
None	

Returns

The return value indicates the current Engine Manager library error condition, if one exists; otherwise, the return value is NULL. DSGetEngineErrorStatus should be checked first after DSPower-HWLib or DSPower-SPLib calls which require an engine handle and which return NULL, followed by DSGetEngMgrErrorStatus (see C/C++ source code example below).

Comments

The Engine Manager error handling scheme is straightforward. Most calls return a non-zero value on success, and a zero (NULL) on failure. When calls into Engine Manager fail, there are two levels of error possibilities: the library itself (or "base level"), and engine-related. Engine-related error conditions are returned by the

DSGetEngineErrorStatus call, while base-level error conditions are returned by DSGetEngMgrErrorStatus.

A single, base-level current error condition variable is maintained internally in the Engine Manager library. If an error condition occurs but is not retrieved, and further calls are made which cause *another* base-level library error condition, DSGetEngMgrErrorStatus returns only the most recent error condition.

Engine Manager also maintains an "error/status listbox" which can be made visible using the DSShowEngMgrStatusWindow call. This status window shows a history of error conditions from all DSPower series library calls.

For more information, see Signalogic DSP Software Users Guide sections 7.5.1, Internal Error Handling in Engine Manager, 7.5.2, Engine Error Handling, and 7.5.3, Error/Status Window in Engine Manager.

C/C++ Example

Below is an example showing error investigation in case of failure of a DSAssignBoard call, which is located inside Hardware Library:

```
int nErrorType;

if (!DSAssignBoard(hEngine, szBoardDesignator, uBusType, uIOBaseAddr,
uMemBaseAddr)) {

    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetEngMgrErrorStatus();
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
```

Note that the DSGetEngineErrorStatus function is used first, since a valid engine handle was involved in the function call which returned NULL. Otherwise, for calls such as DSEngineOpen which do not require an engine handle, DSGetEngMgrErrorStatus should be checked first. Note also that it is possible for error conditions to be overwritten if consecutive function calls are made without error-checking. See **Comments** above.

See Also

DSGetEngineErrorStatus

DSGetHWMgrErrorStatus

DSGetHWLibErrorStatus

DSEngineOpen

Related Users Guide Topics

section 7.5 Error Handling in Engine Manager

section 7.5.1 Internal Error Handling in Engine Manager

section 7.5.2 Engine Error Handling

section 7.5.3 Error/Status Window in Engine Manager

DSHideEngMgrStatusWindow

void DSHideEngMgrStatusWindow ( )

The DSHideEngMgrStatusWindow function hides the Engine Manager error/status listbox window, if not already hidden. The window is not destroyed, nor is its current contents and text display altered.

Parameter	Description
None	

Returns

No return value.

Comments

At least one call to DSAddEngMgrStatusLine must occur before the error/status listbox is created.

By default, the error/status listbox is not visible. C/C++, Visual Basic, MATLAB, and LabView programs using Engine Manager (part of DSPower-HWLib and DSPower-SPLib software) must call DSShowEngMgrStatusWindow at some point to make the listbox visible.

C/C++ Example

```
if (MessageBox(NULL, "hide error/status window?", "??", MB_YESNO) == IDYES) {

    DSHideEngMgrStatusWindow();
}
```

See Also

DSAddEngMgrStatusLine

DSShowEngMgrStatusWindow

Related Users Guide Topics

section 7.5.1 Internal Error Handling in Engine Manager

section 7.5.2 Engine Error Handling

section 7.5.3 Error/Status Window in Engine Manager

DSRegisterEngineMsgWnd

UINT DSRegisterEngineMsgWnd (hEngine,
uMsgType,
hwnd)

```
HENGINE  hEngine;      // engine handle
UINT  uMsgType;        // message type(s)
HWND  hwnd;            // handle of window to register
```

The DSRegisterEngineMsgWnd function registers a window or control handle to receive one or more types of callback messages. A combination of messages can be specified using the pre-defined constants listed below.

Parameter	Description
hEngine	engine handle which was previously obtained via call to DSEngineOpen
uMsgType	one or message types for which hwnd should be registered, as defined below:

Value	Meaning
DS_REMW_SETENGINESTATUSMSG	receive engine status messages
DS_REMW_SETPROGSTATUSMSG	receive engine program and function status messages
DS_REMW_SETDSPDATARDYMSG	receive buffer ready and flag ready messages
DS_REMW_SETSTATISTICSMSG	receive statistic and parameter update messages
DS_REMW_SETBUFNUMMSG	receive buffer number update messages
DS_REMW_SETLEVELMSG	receive data level messages
DS_REMW_SETALLMSG	receive all types of callback messages

hwnd window or control handle to be registered

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid, the window handle was invalid, or one or more message types could not be registered. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines.

Comments

See section 6.1, Library Messages, for information about callback messages which can be received by application programs. The enmgr.h include file supplied with DSPower-HWLib and DSPower-SPLib software lists all callback message types.

C/C++ Example

```
if (!DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETENGINESTATUSMSG |
DS_REMW_SETDSPDATARDYMSG, hwnd)) {

    MessageBox(NULL, "error in DSRegisterEngineMsgWnd", "", MB_OK);
}
```

See Also

DSUnregisterEngineMsgWnd

DSWaitForBuffer

DSWaitForFlag

DSExecEngineFunction

Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Status Messages from DSP Engines

Program Status Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

Update Messages from DSP Engines

DSSendEngineCommand

UINT DSSendEngineCommand (hEngine,
nCommand)
uMode)

```
HENGINE  hEngine;        // engine handle
short int  nCommand;     // command to send
UINT  uMode;             // mode of execution
```

The DSSendEngineCommand function sends commands directly to the specified DSP Engine. Commands can be pre-defined constants defined below, or ASCII or user-defined values as needed.

Parameter	Description
hEngine	engine handle which was previously obtained via call to DSEngineOpen
nCommand	command to send to engine; possible pre-defined command values are listed below:

Value	Meaning
DS_SEC_ABORT	instruct engine to abort (quit)
DS_SEC_IDLE	instruct engine to enter idle mode
DS_SEC_PAUSE	suspend engine (only valid for DOS engines)
DS_SEC_RESUME	resume engine (only valid for DOS engines)

uMode — mode in which command should be sent; one of the possible pre-defined values listed below should be used:

Value	Meaning
DS_EEF_ASYNC	send command asynchronously with respect to calling application (calling application continues to execute simultaneously as engine executes command); this is the default used if uMode is left NULL
DS_EEF_SYNC	send command synchronously with respect to calling application (calling application waits for engine to complete command)

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid, or the command was not sent. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines.

Comments

Pre-defined nCommand constants described above are translated by Engine Manager into correct DSP Engine commands. ASCII printable or control characters are sent to the engine without modification. For example the calling application could send command strings to MATLAB by using the DSSendEngineCommand function in a loop which sent each character of the string, and replaced the terminating NULL with a 0x0a value (linefeed).

Pre-defined nCommand constants listed above are encoded as negative values; to ensure that a calling application does not sent commands which will be translated by Engine Manager, consult the enmgr.h include file.

Either synchronous or asynchronous command execution can be specified using the uMode parameter. Synchronous operation means the calling application waits for the engine to execute the command; asynchronous operation means the calling application continues to execute simultaneously.

NOTE: Currently, a uMode value of DS_EEF_ASYNC is only active for the DS_SEC_IDLE and DS_SEC_ABORT commands, which initialize a wait for a WM_DSPENGINE_PROGSTATUS message with DS_PS_PROGDONE parameter.

C/C++ Example 1

```
DSSendEngineCommand(hEngine, DS_SEC_IDLE);  // put engine in idle mode
```

C/C++ Example 2

```
char szCommand[100];
int i;

for (i=0; i<_fstrlen(szCommand); i++) {

    // send command text string

    DSSendEngineCommand(hEngine, szCommand[i], NULL);  // asynchronous
}
DSSendEngineCommand(hEngine, 0x0a, NULL);  // send Enter key
```

See Also

DSExecEngineFunction

WM_DSPENGINE_PROGSTATUS

Related Users Guide Topics

section 7.3 Executing Functions in DSP Engines

DSSetEngineTemplatePath

UINT DSSetEngineTemplatePath (hEngine,
pszTemplatePath)

HENGINE hEngine; // engine handle

```
LPCSTR  pszTemplatePath;          // path for engine template files
```

The DSSetEngineTemplatePath function sets the path used by the specified engine for locating template files.

Parameter	Description
hEngine	engine handle; should be obtained using call to DSEngineOpen
pszTemplatePath	pointer to null-terminated string containing path of template files which may need to be accessed in creating engine programs for high-level functions

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid, or the path could not be set. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines.

Comments

Template files are source code files in the native language of the DSP Engine, and are used as the basis for creating executable engine programs. Depending on the DSP Engine, high-level functions such as DSAcquireWvfrmFile, DSGenerateWvfrmFile, and DSExecEngineFunction may be implemented using this technique.

C/C++ Example

```
DSSetEngineTemplatePath(hEngine, "c:\\dspower\\syslib");
```

See Also

DSAcquireWvfrmFile

DSGenerateWvfrmFile

DSExecEngineFunction

Related Users Guide Topics

section 4.5 Acquiring Data Using DSPower-HWLib Software

section 7.3 Executing Functions in DSP Engines

DSSetEngineWaveformPath

UINT DSSetEngineWaveformPath (hEngine,
pszWaveformPath)

HENGINE hEngine; // engine handle

LPCSTR pszWaveformPath; // path for engine waveform files

The DSSetEngineWaveformPath function sets the path used by the specified engine for locating waveform files.

Parameter	Description
hEngine	engine handle; should be obtained using call to DSEngineOpen
pszWaveformPath	pointer to null-terminated string containing path of waveform files which might be accessed by the DSP Engine

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid, or the path could not be set. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines.

Comments

Waveform files can be in Hypersignal (.tim, .frq, etc.), MATLAB (.mat), Microsoft (.wav), or various acoustic/audio waveform file formats. For more information, see chapter 4, Waveform File Formats.

C/C++ Example

```
DSSetEngineWaveformPath(hEngine, "c:\\hsmacro\\waveform");
```

See Also

DSAcquireWvfrmFile

DSGenerateWvfrmFile

Related Users Guide Topics

section 4.5 Acquiring Data Using DSPower-HWLib Software

section 7.3 Executing Functions in DSP Engines

DSShowEngMgrStatusWindow

void DSShowEngMgrStatusWindow ( )

The DSShowEngMgrStatusWindow function makes the Engine Manager error/status listbox window visible, if not already visible. The window's current contents and text display are not altered.

Parameter	Description
None	

Returns

No return value.

Comments

At least one call to DSAddEngMgrStatusLine must occur before the error/status listbox is created.

By default, the error/status listbox is not visible. C/C++, Visual Basic, MATLAB, and LabView programs using Engine Manager (part of DSPower-HWLib and DSPower-SPLib software) must call DSShowEngMgrStatusWindow at some point to make the listbox visible.

C/C++ Example

```
DSAddEngMgrStatusLine("add this line before showing window");
DSShowEngMgrStatusWindow();
```

See Also

DSAddEngMgrStatusLine

Related Users Guide Topics

section 7.5.1 Internal Error Handling in Engine Manager

section 7.5.2 Engine Error Handling

section 7.5.3 Error/Status Window in Engine Manager

DSUnregisterEngineMsgWnd

UINT DSUnregisterEngineMsgWnd (hEngine,
uMsgType)

```
HENGINE hEngine;        // engine handle
UINT uMsgType;          // message type(s)
```

The DSUnregisterEngineMsgWnd function unregisters the current window(s) associated with one or more types of callback messages, after which the specified callback message types are no longer sent to the window(s). A combination of message types can be specified using the pre-defined constants listed below.

Parameter	Description
hEngine	engine handle which was previously obtained using call to DSEngineOpen
uMsgType	one or more message types for which the current registered window(s) should be unregistered, as defined below:

Value	Meaning
DS_REMW_SETENGINESTATUSMSG	unregister window currently receiving engine status messages
DS_REMW_SETPROGSTATUSMSG	unregister window currently receiving engine program and function status messages
DS_REMW_SETDSPDATARDYMSG	unregister window

	currently receiving buffer ready and flag ready messages
DS_REMW_SETSTATISTICSMSG	unregister window currently receiving statistic and parameter update messages
DS_REMW_SETBUFNUMMSG	unregister window currently receiving buffer number update messages
DS_REMW_SETLEVELMSG	unregister window currently receiving data level messages
DS_REMW_SETALLMSG	unregister all windows currently receiving any callback messages

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid or one or more message types could not be unregistered. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines.

Comments

See section 6.1, Library Messages, for information about callback messages which can be received by application programs. The enmgr.h include file supplied with DSPower-HWLib and DSPower-SPLib software lists all callback message types.

C/C++ Example

```
DSUnregisterEngineMsgWnd(hEngine, DS_REMW_SETENGINESTATUSMSG |
DS_REMW_SETDSPDATARDYMSG);
```

See Also

DSRegisterEngineMsgWnd

DSWaitForBuffer

DSWaitForFlag

DSExecEngineFunction

Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Status Messages from DSP Engines

Program Status Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

Update Messages from DSP Engines

2.2.2 Hardware Manager Functions

[@k functions, Hardware Manager, hwmgr.dll, hwmgr32.dll]

Below are listed functions in the Hardware Manager API. For information on how to include Hardware Manager (hwmgr.dll or hwmgr32.dll) in C/C++, Visual Basic, MATLAB, and LabView projects, see Signalogic DSP Software Users Guide section 8.1, Hardware Manager Access.

Hardware Manager Functions

DSGetHWMgrCurVal

DSGetHWMgrEntry

DSSetHWMgrEntry

DSGetHWMgrEntryIndex

DSGetHWMgrErrorStatus

DSGetHWMgrNumEntries

DSGetHWMgrVar

DSReadHWSetupFile

DSSetHWMgrCurVal

DSSetHWMgrVar

DSShowHardwareSelectorDlg

DSWriteHWSetupFile

DSGetHWMgrCurVal

UINT DSGetHWMgrCurVal (pszBoardDesignator)

LPSTR pszBoardDesignator; // pointer to board designator string

The DSGetHWMgrCurVal function retrieves the board designator for the current hardware value selected in the Hardware Selector.

Parameter	Description
pszBoardDesignator	pointer to string variable which will contain a null-terminated board designator string upon return

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the board designator string for the current hardware value in the Hardware Selector could not be retrieved. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

The string variable pointed to by pszBoardDesignator is assumed to allow enough space for a return value (no space validation of any kind is performed). At least 50 characters is suggested, although board designators are normally no longer than 10-12 characters. For more information about board designators, see Board Designator Field in Hardware Setup File, in Signalogic DSP Software Users Guide section 8.2.2, DSP/Analog Hardware Configuration.

For more information about setting the current hardware value (including the current Board Designator field) in the Hardware Selector, see Signalogic DSP Software Users Guide section 8.2.1, Selecting DSP/Analog Hardware Types.

For more information about hardware values, see Signalogic DSP Software Users Guide section 8.2.3, Hardware Values and Variables.

C/C++ Example

The example below shows a DSGetHWMgrCurVal function call being used to retrieve the current Board Designator from the Hardware Selector, and then using this value to assign a board handle.

```
char szBoardDesignator[50];
int nErrorType;
HBOARD hBoard;

if (!DSGetHWMgrCurVal(szBoardDesignator)) {

    nErrorType = DSGetHWMgrErrorStatus();
    //  do error processing...
}
else hBoard = DSAssignBoard(hEngine, szBoardDesignator, NULL, NULL, NULL);
```

See Also

DSSetHWMgrCurVal

DSAssignBoard

DSGetHWMgrEntryIndex

DSShowHardwareSelectorDlg

Related Users Guide Topics

8.2.1 Selecting DSP/Analog Hardware Types

8.2.3 Hardware Values and Variables

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

Board Designator, in section 8.2.2, DSP/Analog Hardware Configuration

Board Designator Field in Hardware Setup File, in section 8.2.2, DSP/Analog Hardware Configuration

DSGetHWMgrEntry

UINT DSGetHWMgrEntry (nIndex,
pHWList)

short int nIndex; // index into Hardware Setup File

PHWLIST pHWList; // pointer to HWLISTENTRY structure

The DSGetHWMgrEntry function retrieves an entry from the Hardware Setup File, storing it in a HWLISTENTRY structure.

Parameter	Description
nIndex	index specifying which Hardware Setup File entry to retrieve
pHWList	pointer to HWLISTENTRY structure which will contain entry from Hardware Setup File

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating the Hardware Setup File index was out of range. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

The Hardware Setup File is initially read into global memory (allocated by Hardware Manager) from disk file upon the first instance of a call into Hardware Manager requiring data DSP/analog hardware configuration data. This occurs automatically, and includes the DSShowHardwareSelectorDlg call.

It is possible to obtain DSP/analog hardware configuration data from Hardware Manager-allocated global memory by calling the DSGetHWMgrEntry function with a pointer to a HWLISTENTRY structure. If the information is modified and needs to be saved, DSSetHWMgrEntry and DSWriteHWSetupFile calls should be made. Note that if the Hardware Selector is invoked by the operator, and DSP/analog hardware configuration data subsequently changed followed by pushing the "OK" button, DSWriteHWSetupFile is executed automatically.

For more information, see Signalogic DSP Software Users Guide sections 8.3, Hardware Setup File, 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure (in section 8.3.2).

C/C++ Example

Below is an example of using the DSGetHWMgrEntryIndex and DSGetHWMgrEntry calls to obtain Hardware Setup File information:

```
int nErrorType, nIndex;
```

```
HWLISTENTRY HWListEntry;

_fmemset(&HWListEntry, 0, sizeof(HWLISTENTRY));
nIndex = DSGetHWMgrEntryIndex("IIC31");
if (!nIndex || !DSGetHWMgrEntry(&HWListEntry, nIndex)) {

    nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
```

Note that the HWListEntry structure is initialized to zero before using, causing most structure elements to be set their default state. Note also that DSGetHWMgrEntryIndex can return NULL if the Hardware Setup File entry specified by this function's pszBoardDesignator input parameter is not found.

See Also

HWLISTENTRY

DSGetHWMgrEntryIndex

DSGetHWMgrNumEntries

DSSetHWMgrEntry

DSReadHWSetupFile

DSWriteHWSetupFile

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

DSGetHWMgrEntryIndex

UINT DSGetHWMgrEntryIndex **(pszBoardDesignator),**

LPCSTR pszBoardDesignator; // pointer to board designator string

The DSGetHWMgrEntryIndex function retrieves the index of the Hardware Setup File entry matching the specified Board Designator.

Parameter	Description
pszBoardDesignator	pointer to null-terminated board designator string for which to find a Hardware Setup File index

Returns

The return value is NULL if the specified board designator string does not match any of the entries in the Hardware Setup File, or if there is a file-access problem. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

Index values returned by DSGetHWMgrEntryIndex can range from 0 to the current total number of Hardware Setup File entries – 1, inclusively; the upper value can be determined by calling DSGetHWMgrNumEntries.

The DSGetHWMgrEntryIndex function can be combined with DSGetHWMgrEntry and DSSetHWMgrEntry to modify DSP/analog hardware configuration data in the Hardware Setup File. The Reference Guide descriptions for the latter two functions contain comments and detailed source code examples which provide more information about this process.

For more information about the arrangement, contents, and access of the Hardware Setup File, see Signalogic DSP Software Users Guide sections 8.3, Hardware Setup File, 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure (in section 8.3.2).

C/C++ Example

Below is an example of using the DSGetHWMgrEntryIndex function to look for the index of an entry in the Hardware Setup File matching a particular Board Designator string:

```
int nErrorType, nIndex;

nIndex = DSGetHWMgrEntryIndex("ARI3210");
if (!nIndex) {
```

```
    nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
```

Note that DSGetHWMgrEntryIndex returns NULL if the Hardware Setup File entry specified by the function's Board Designator string input parameter is not found. For more complete C/C++ source code examples, see Reference Guide descriptions for DSGetHWMgrEntry and DSSetHWMgrEntry.

See Also

DSGetHWMgrNumEntries

DSGetHWMgrEntry

DSSetHWMgrEntry

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

Board Designator Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSGetHWMgrErrorStatus

UINT DSGetHWMgrErrorStatus ( )

The DSGetHWMgrErrorStatus function retrieves the current base-level error status for the Hardware Manager library (hwmgr.dll or hwmgr32.dll).

Parameter	Description
None	

Returns

The return value indicates the current Hardware Manager library error condition, if one exists; otherwise, the return value is NULL. See C/C++ source code example below.

Comments

The Hardware Manager error handling scheme is straightforward. Most calls return a non-zero value on success, and a zero (NULL) on failure. When calls into Hardware Manager fail, the DSGetHWMgrErrorStatus call should be invoked.

A single, base-level current error condition variable is maintained internally in the Hardware Manager library. If an error condition occurs but is not retrieved, and further calls are made which cause *another* base-level library error condition, DSGetHWMgrErrorStatus returns only the most recent error condition.

Note that Engine Manager maintains an "error/status listbox" which can be made visible using the DSShowEngMgrStatusWindow call. This status window shows a history of error conditions from all DSPower series library calls, including Hardware Manager.

For more information, see Signalogic DSP Software Users Guide sections 8.4, Error Handling in Hardware Manager, and 8.4.1, Internal Error Handling in Hardware Manager. See also section 7.5.3, Error/Status Window in Engine Manager.

C/C++ Example

Below is an example showing error investigation in case of failure of a DSGetHWMgrEntryIndex call:

```
int nErrorType, Index;

Index = DSGetHWMgrEntryIndex("IIC31");
if (!Index) {

    nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
```

See Also

DSGetEngMgrErrorStatus

DSGetEngineErrorStatus

DSGetHWLibErrorStatus

Related Users Guide Topics

section 8.4 Error Handling in Hardware Manager

section 8.4.1 Internal Error Handling in Hardware Manager

section 7.5.3 Error/Status Window in Engine Manager

DSGetHWMgrNumEntries

UINT DSGetHWMgrNumEntries ( ),

The DSGetHWMgrNumEntries function retrieves the total number of current entries in the Hardware Setup File.

Parameter	Description
None	

Returns

The return value is NULL if the Hardware Setup File has not yet been read or cannot be read into Hardware Manager memory, or there are no valid entries in the file. Otherwise, the return value indicates the number of entries found in the Hardware Setup File. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

The DSGetHWMgrNumEntries function can be combined with DSGetHWMgrEntryIndex, DSGetHWMgrEntry, and DSSetHWMgrEntry to modify DSP/analog hardware configuration data in the Hardware Setup File. The Reference Guide descriptions for the latter two functions contain comments and detailed source code examples which provide more information about this process.

The DSGetHWMgrEntryIndex function can be used to retrieve the index of a Hardware Setup File entry matching a specified board designator string. This function returns values ranging from 0 to DSGetHWMgrNumEntries – 1, inclusively.

For more information about the arrangement, contents, and access of the Hardware Setup File, see Signalogic DSP Software Users Guide

sections 8.3, Hardware Setup File, 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure (in section 8.3.2).

C/C++ Example

Below is an example of using the DSGetHWMgrNumEntries function to create a loop in which Hardware Setup File entries with certain configuration data are searched for:

```
int nErrorType, nNumEntries, i;
HWLISTENTRY HWListEntry;
WORD wMemBaseAddr;

nNumEntries = DSGetHWMgrNumEntries();
if (!nNumEntries) {

    nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
else for (i=0; i<nNumEntries; i++) {  // loop through all entries

    _fmemset(&HWListEntry, 0, sizeof(HWLISTENTRY));

    if (!DSGetHWMgrEntry(&HWListEntry, i)) {  // get the entry

        nErrorType = DSGetHWLibErrorStatus();
        ...do error processing...
        break; // exit loop
    }
    else {

        sscanf(HWListEntry.MemBaseAddr, %x, &wMemBaseAddr);

        if (wMemBaseAddr >= 0xD000  &&  wMemBaseAddr <= 0xEFFF) {

            // process memory-mapped hardware in region D000H - EFFFH
        }
    }
}
```

For additional C/C++ source code examples, see Reference Guide descriptions for DSGetHWMgrEntry and DSSetHWMgrEntry.

See Also

DSGetHWMgrEntry

DSSetHWMgrEntry

DSGetHWMgrEntryIndex

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

Board Designator Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSGetHWMgrVar

UINT DSGetHWMgrVar (wVarIndex,
pszBoardDesignator)

```
WORD  wVarIndex;          // index of hardware variable
LPSTR  pszBoardDesignator; // pointer to board designator string
```

The DSGetHWMgrVar function retrieves the board designator string for the specified hardware variable in the Hardware Selector.

Parameter	Description
wVarIndex	index of hardware variable for which to retrieve board designator string; should be one of the pre-defined hardware variable constants listed below:

Value	Meaning
RTE	Real-Time Engine
SA	Spectrum Analyzer
ACC	DSP Accelerator
DTR	Digital Tape Recorder
ACQ	Data Acquisition (A/D conversion, waveform file record)
CSG	Continuous Signal Generation (periodic function generation and arbitrary signal synthesis)
SR	Stimulus & Response
PB	Playback (waveform file output)

	DO	Digital Oscilloscope
	UD1	User-Defined 1
	UD2	User-Defined 2
	UD3	User-Defined 3
	UD4	User-Defined 4
	UD5	User-Defined 5
pszBoardDesignator	pointer to string variable which will contain a null-terminated board designator string upon return	

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating the hardware variable was out of range (greater than or equal to MAXBOARDVARS), or the board designator string could not be retrieved for the specified hardware variable in the Hardware Selector. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

Hardware variables appear in a subdialog box that is invoked from the Hardware Selector by pushing the button marked "Set Hardware Variables". Currently, this button is enabled only when the Hardware Selector is displayed from DSPower-Block Diagram software. However, hardware variables can still be set and retrieved at any time by user-defined programs using the DSGetHWMgrVar and DSSetHWMgrVar functions in Hardware Manager.

The string variable pointed to by pszBoardDesignator is assumed to allow enough space for a return value (no space validation of any kind is performed). At least 50 characters is suggested, although board designators are normally no longer than 10-12 characters. For more information about board designators, see Board Designator Field in Hardware Setup File, in Signalogic DSP Software Users Guide section 8.3.1, Hardware Setup File Fields.

For more information about hardware variables, see Signalogic DSP Software Users Guide section 8.2.3, Hardware Values and Variables.

C/C++ Example

The example below shows a DSGetHWMgrVar function call being used to retrieve the board designator string for the data acquisition hardware variable from the Hardware Selector, and then using this value to assign a board handle.

```
char szBoardDesignator[50];
int nErrorType;
HBOARD hBoard;

if (!DSGetHWMgrVar(ACQ, szBoardDesignator)) {

      nErrorType = DSGetHWMgrErrorStatus();
      //  do error processing...
}
else hBoard = DSAssignBoard(hEngine, szBoardDesignator, NULL, NULL, NULL);
```

See Also

DSSetHWMgrVar

DSGetHWMgrEntryIndex

DSShowHardwareSelectorDlg

DSGetHWMgrCurVal

DSSetHWMgrCurVal

DSAssignBoard

Related Users Guide Topics

section 8.2.3 Hardware Values and Variables

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

Board Designator, in section 8.2.2, DSP/Analog Hardware Configuration

Board Designator Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSGetHWSetupFile

UINT DSReadHWSetupFile (pszSetupFile)

LPSTR pszSetupFile;	// pointer to returned value of hardware setup file

The DSGetHWSetupFile function retrieves the current Hardware Setup File for the calling application. Note that no actual setup file access is made (see DSReadHWSetupFile description).

Parameter	Description
pszSetupFile	pointer to memory area sufficient to receive null-terminated Hardware Setup File string, which will contain full path information

Returns

The return value is NULL if the pszSetupFile parameter is NULL.

Comments

The current Hardware Setup File in use by a calling application can be specified by making DSSetHWSetupFile or DSReadHWSetupFile calls.

Currently, the default Hardware Setup File is named hwsetup.lst, and the default location of this file is the location of Hardware Manager (hwmgr.dll or hwmgr32.dll).

Hardware Manager keeps track of different setup files on an application-specific basis. Different applications may use different setup files or the same setup file. Also, different setup files may be open and updated at the same time. The current maximum number of applications which may be using Hardware Manager at one time is eight (8).

For more information, see Signalogic DSP Software Users Guide section 8.3, Hardware Setup File, section 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure in section 8.3.2.

See Also

DSReadHWSetupFile

DSSetHWSetupFile

DSWriteHWSetupFile

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

DSReadHWSetupFile

HGLOBAL DSReadHWSetupFile (pszSetupFile)

LPCSTR pszSetupFile; // pointer to hardware setup file

The DSReadHWSetupFile function allocates global memory and reads HWLISTENTRY entries from the specified Hardware Setup File (containing DSP/analog hardware configuration data) into this memory A NULL value for pszSetupFile causes the default "hwsetup.lst" setup file, or previous setup file specified by DSSetHWSetupFile, to be read.

Parameter	Description
pszSetupFile	pointer to null-terminated Hardware Setup File string; this string should specify the location of the setup file, including path information if needed; without path information, the default location used by Hardware Manager (location of hwmgr.dll file or hwmgr32.dll) will be used

Returns

The return value is NULL if either the specified Hardware Setup File could not be located or a problem occurred allocating memory or reading the file. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below). Otherwise, the return value is the global memory handle to a series of HWLISTENTRY records containing DSP/analog hardware configuration data found in the specified setup file.

Comments

If no initial call is made to DSReadHWSetupFile, the default Hardware Setup File is initially read into global memory from disk file upon the first instance of a call into Hardware Manager requiring

DSP/analog hardware configuration data. This occurs automatically, and includes the DSShowHardwareSelectorDlg call.

Memory is allocated as each new entry is read from the Hardware Setup File. If at some point, memory cannot be increased, NULL is returned and a subsequent call to DSGetHWMgrErrorStatus will return DSCOULDNOTINCREASEHWLISTMEM.

Currently, the default Hardware Setup File is named hwsetup.lst, and the default location of this file is the location of Hardware Manager (hwmgr.dll or hwmgr32.dll).

Hardware Manager keeps track of different setup files on an application-specific basis. Different applications may use different setup files or the same setup file. Also, different setup files may be open and updated at the same time. The current maximum number of applications which may be using Hardware Manager at one time is eight (8).

It is possible to obtain and change DSP/analog hardware configuration data programmatically by calling the DSGetHWMgrEntry and DSSetHWMgrEntry functions with a pointer to a HWLISTENTRY structure. In this case, a DSWriteHWSetupFile call should be made if DSP/analog hardware configuration data has been changed, and needs to be saved. Alternatively, setup files may be altered directly using a standard text editor, outside of the Hardware Manager environment. Note that if the Hardware Selector is invoked by the operator, and DSP/analog hardware configuration data subsequently changed followed by pushing the "OK" button, DSWriteHWSetupFile is executed automatically.

If the DSReadHWSetupFile function is successful, the return value is the global memory handle to a series of HWLISTENTRY records containing DSP/analog hardware configuration data. The number of records can be determined by calling DSGetHWMgrNumEntries. The global memory handle is created using the GMEM_SHARED attribute.

NOTE: It is not recommended that calling applications access HWLISTENTRY records directly using the global memory handle returned by DSReadHWSetupFile. Instead, the DSGetHWMgrEntry call should be used.

Note that the current hardware value setting, as well as hardware variable settings, are not stored in the Hardware Setup File. They are

instead stored in a separate private configuration file maintained by Hardware Manager (hwmgr.ini), on a application-independent basis (separate sections inside the configuration file are used).

For more information, see Signalogic DSP Software Users Guide section 8.3, Hardware Setup File, section 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure in section 8.3.2.

C/C++ Example

The example below shows the DSReadHWSetupFile function being called:

```
HGLOBAL hHWList;
int nErrorType;
LPCSTR szSetupFile = NULL;

hHWList = DSReadHWSetupFile(szSetupFile);

if (!hHWList) {

    nErrorType = DSGetHWMgrErrorStatus();
    //  do error processing...
}
```

See Also

DSWriteHWSetupFile

DSGetHWMgrEntry

DSSetHWMgrEntry

DSGetHWMgrNumEntries

DSShowHardwareSelectorDlg

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

DSSetHWMgrCurVal

UINT DSSetHWMgrCurVal (pszBoardDesignator)

```
LPCSTR pszBoardDesignator;          // pointer to board designator string
```

The DSSetHWMgrCurVal function sets the current hardware value in the Hardware Selector to match the specified board designator.

Parameter	Description
pszBoardDesignator	pointer to null-terminated string containing board designator

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the board designator string was invalid or no match could be found in the current Hardware Setup File entries. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source example below).

Comments

For more information about the current hardware value in the Hardware Selector, see Signalogic DSP Software Users Guide section 8.2.1, Selecting DSP/Analog Hardware Types.

For more information about hardware values, see Signalogic DSP Software Users Guide section 8.2.3, Hardware Values and Variables.

For more information about board designators, see Board Designator Field in Hardware Setup File, in Signalogic DSP Software Users Guide section 8.3.1, Hardware Setup File Fields.

C/C++ Example

The example below shows a DSSetHWMgrCurVal function call being used to set the current board designator prior to displaying the Hardware Selector dialog box:

```
char szBoardDesignator[50], szUserBoardDesignator[50];
int nErrorType;

lstrcpy(szBoardDesignator, "TBS56-B");

if (!DSSetHWMgrCurVal(szBoardDesignator)) {
```

```
    nErrorType = DSGetHWMgrErrorStatus();
    //  do error processing...
}
else {

    DSShowHardwareSelectorDlg(hwnd, szUserBoardDesignator); // hwnd is owner

    if (lstrcmpi(szBoardDesignator, szUserBoardDesignator) != 0) {

        // user has made a different choice; process...
    }
}
```

Note that the DSShowHardwareSelectorDlg call also returns a board designator. In the above example, if the user did not specify a new choice while the Hardware Selector was displayed, or pushed the "Cancel" button, then szBoardDesignator and szUserBoardDesignator would end up with the same value.

See Also

DSGetHWMgrCurVal

DSShowHardwareSelectorDlg

DSGetHWMgrEntryIndex

Related Users Guide Topics

section 8.2.1 Selecting DSP/Analog Hardware Types

section 8.2.3 Hardware Values and Variables

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

Board Designator, in section 8.2.2, DSP/Analog Hardware Configuration

Board Designator Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSSetHWMgrEntry

UINT DSSetHWMgrEntry (nIndex,
pHWList)

short int nIndex; // index into Hardware Setup File

```
PHWLIST  pHWList;                    // pointer to HWLISTENTRY structure
```

The DSSetHWMgrEntry function sets an entry in the Hardware Setup File, using information contained in a HWLISTENTRY structure.

Parameter	Description
nIndex	index specifying which Hardware Setup File entry to set
pHWList	pointer to HWLISTENTRY structure containing information with which to set the Hardware Setup File entry

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating the nIndex parameter was out of range (greater than the current value returned by DSGetHWMgrNumEntries). For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

If nIndex is equal to the value returned by DSGetHWMgrNumEntries (i.e. nIndex is one more than the index of the last existing entry), the entry specified by pHWList will be added, rather than replacing an existing entry.

The Hardware Setup File is initially read into global memory (allocated by Hardware Manager) from disk file upon the first instance of a call into Hardware Manager requiring data DSP/analog hardware configuration data. This occurs automatically, and includes the DSShowHardwareSelectorDlg call.

It is possible to obtain and set DSP/analog hardware configuration data from Hardware Manager-allocated global memory by calling the DSGetHWMgrEntry and DSSetHWMgrEntry functions with a pointer to a HWLISTENTRY structure. If the information is modified and needs to be saved, a DSWriteHWSetupFile call should be made. Note that if the Hardware Selector is invoked by the operator, and DSP/analog hardware configuration data subsequently changed followed by pushing the "OK" button, DSWriteHWSetupFile is executed automatically.

For more information, see Signalogic DSP Software Users Guide sections 8.3, Hardware Setup File, 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure (in section 8.3.2).

C/C++ Example

Below is an example of using the DSGetHWMgrEntry call to modify Hardware Setup File information, which has previously been retrieved using DSGetHWMgrEntryIndex and DSGetHWMgrEntry:

```
int nErrorType, nIndex;
HWLISTENTRY HWListEntry;

nIndex = DSGetHWMgrEntryIndex("IIC31");
if (!nIndex || !DSGetHWMgrEntry(&HWListEntry, nIndex)) {

    nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
else {

    // make changes to HWLISTENTRY information

    lstrcpy(HWListEntry.IOBaseAddr, "0x340");
    lstrcpy(HWListEntry.VoltageRanges, "1.4,1.8");
    HWListEntry.BusType = DS_AB_PCAT;

    if (!DSSetHWMgrEntry(&HWListEntry, nIndex)) {

        nErrorType = DSGetHWLibErrorStatus();
        ...do error processing...
    }
}
```

Note that the HWListEntry structure is initialized to zero before using, causing most structure elements to be set their default state. Note also that DSGetHWMgrEntryIndex can return NULL if the Hardware Setup File entry specified by the function's board designator string input parameter is not found.

See Also

HWLISTENTRY

DSGetHWMgrEntryIndex

DSGetHWMgrNumEntries

DSGetHWMgrEntry

DSReadHWSetupFile

DSWriteHWSetupFile

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

DSSetHWMgrVar

UINT DSSetHWMgrVar (wVarIndex,
pszBoardDesignator)

```
WORD  wVarIndex;            // index of hardware variable
LPCSTR  pszBoardDesignator; // pointer to board designator string
```

The DSSetHWMgrVar function sets the board designator string for the specified hardware variable in the Hardware Selector.

Parameter	Description
wVarIndex	index of hardware variable for which to set board designator string; should be one of the pre-defined hardware variable constants listed below:

Value	Meaning
RTE	Real-Time Engine
SA	Spectrum Analyzer
ACC	DSP Accelerator
DTR	Digital Tape Recorder
ACQ	Data Acquisition (A/D conversion, waveform file record)
CSG	Continuous Signal Generation (periodic function generation and arbitrary signal synthesis)
SR	Stimulus & Response

	PB	Playback (waveform file output)
	DO	Digital Oscilloscope
	UD1	User-Defined 1
	UD2	User-Defined 2
	UD3	User-Defined 3
	UD4	User-Defined 4
	UD5	User-Defined 5
pszBoardDesignator	pointer to string variable containing a null-terminated board designator string	

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating the hardware variable was out of range (greater than or equal to MAXBOARDVARS), indicating that the Board Designator string was invalid or no match could be found in the current Hardware Setup File entries. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

Hardware variables appear in a subdialog box that is invoked from the Hardware Selector by pushing the button marked "Set Hardware Variables". Currently, this button is enabled only when the Hardware Selector is displayed from DSPower-Block Diagram software. However, hardware variables can still be set and retrieved at any time by user-defined programs using the DSGetHWMgrVar and DSSetHWMgrVar functions in Hardware Manager.

The string variable pointed to by pszBoardDesignator is assumed to allow enough space for a return value (no space validation of any kind is performed). At least 50 characters is suggested, although board designators are normally no longer than 10-12 characters. For more information about board designators, see Board Designator Field in Hardware Setup File, in Signalogic DSP Software Users Guide section 8.3.1, Hardware Setup File Fields.

For more information about hardware variables, see Signalogic DSP Software Users Guide section 8.2.3, Hardware Values and Variables.

C/C++ Example

The example below shows a DSSetHWMgrVar function call being used to set the Board Designator string for the continuous signal generation hardware variable in the Hardware Selector:

```
int nErrorType;

if (!DSSetHWMgrVar(CSG, "NAT32C")) {

    nErrorType = DSGetHWMgrErrorStatus();
    //  do error processing...
}
```

See Also

DSGetHWMgrVar

DSShowHardwareSelectorDlg

DSGetHWMgrCurVal

DSSetHWMgrCurVal

DSGetHWMgrEntryIndex

Related Users Guide Topics

section 8.2.3 Hardware Values and Variables

Board Designator, in section 8.2.2, DSP/Analog Hardware Configuration

Board Designator Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSSetHWSetupFile

UINT DSReadHWSetupFile (pszSetupFile)

LPCSTR pszSetupFile; // pointer to hardware setup file

The DSSetHWSetupFile function sets the current Hardware Setup File for the calling application. Note that no actual setup file access is made (see DSReadHWSetupFile description).

Parameter	Description
pszSetupFile	pointer to null-terminated Hardware Setup File string; this string should specify the location of the setup file, including path information if needed; without path information, the default location used by Hardware Manager (location of hwmgr.dll file or hwmgr32.dll) will be used

Returns

The return value is NULL if the pszSetupFile parameter is NULL.

Comments

The current Hardware Setup File in use by a calling application can also be specified by making a DSReadHWSetupFile call.

Currently, the default Hardware Setup File is named hwsetup.lst, and the default location of this file is the location of Hardware Manager (hwmgr.dll or hwmgr32.dll).

Hardware Manager keeps track of different setup files on an application-specific basis. Different applications may use different setup files or the same setup file. Also, different setup files may be open and updated at the same time. The current maximum number of applications which may be using Hardware Manager at one time is eight (8).

For more information, see Signalogic DSP Software Users Guide section 8.3, Hardware Setup File, section 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure in section 8.3.2.

See Also

DSGetHWSetupFile

DSReadHWSetupFile

DSWriteHWSetupFile

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

DSShowHardwareSelectorDlg

long DSShowHardwareSelectorDlg (hwnd, pszBoardDesignator)

```
HWND  hwnd;                  // handle of owner window
LPSTR  pszBoardDesignator;   // pointer to board designator string
```

The DSShowHardwareSelectorDlg function activates and displays the Hardware Selector dialog box.

Parameter	Description
hwnd	handle of owner window for Hardware Selector dialog box, if any
pszBoardDesignator	pointer to string variable which will contain a null-terminated board designator string upon return

Returns

The return value is –1 if the Hardware Selector cannot be displayed, or if a problem occurs reading the Hardware Setup File. Otherwise, the low word of the return value contains the result of a DialogBoxParam call, and the high word contains a constant indicating whether configuration data for one or more hardware values was changed, or any hardware variables were changed, or both.

Typically, the low word of the return value will contain the identifier of the button pushed to close the dialog box, usually IDOK or IDCANCEL. The high word of the return value will have bit 0 set if the configuration data for a hardware value was changed, and bit 1 set if any hardware variables were changed. For further error checking if a –1 value should be returned, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

If not already read, the Hardware Setup File is read into global memory (allocated by Hardware Manager) from disk file when the DSShowHardwareSelectorDlg call is invoked.. If DSP/analog hardware configuration data for one or more hardware values is changed by the operator while the Hardware Selector dialog box is displayed (and subsequently followed by pushing the "OK" button), a DSWriteHWSetupFile call is executed automatically before DSShowHardwareSelectorDlg returns.

It is also possible to obtain and set DSP/analog hardware configuration data from Hardware Manager-allocated global memory by calling the DSGetHWMgrEntry and DSSetHWMgrEntry functions with a pointer to a HWLISTENTRY structure. If the information is modified and needs to be saved, a DSWriteHWSetupFile call should also be made

For more information about the arrangement, contents, and means of access of the Hardware Setup File, see Signalogic DSP Software Users Guide sections 8.3, Hardware Setup File, 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure (in section 8.3.2).

The string variable pointed to by pszBoardDesignator is assumed to allow enough space for a return value (no space validation of any kind is performed). At least 50 characters is suggested, although board designators are normally no longer than 10-12 characters. For more information about board designators, see Board Designator Field in Hardware Setup File, in Signalogic DSP Software Users Guide section 8.3.1, Hardware Setup File Fields.

The DSShowHardwareSelectorDlg invokes a DialogBoxParam call with "HardwareSetup" as the resource name, and pszBoardDesignator as the extra parameter information.

C/C++ Example

Below is an example showing the DSShowHardwareSelectorDlg call being used to prepare for a Hardware Library DSAssignBoard call:

```
char szBoardDesignator[50];
HENGINE hEngine;

if (DSShowHardwareSelectorDlg(NULL, szBoardDesignator) != IDCANCEL) {

    // assign board handle:  engine, bd designator, bus type, IO & mem base addrs
    hBoard = DSAssignBoard(hEngine, szBoardDesignator, NULL, NULL, NULL);

    // initialize the board; make sure it's installed and responded
```

```
    if (!DSInitBoard(hBoard)) {

        itoa(DSGetEngineErrorStatus(hEngine), tmpstr, 10);
        MessageBox(NULL, tmpstr, "Test Prog:  DSInitBoard failed", MB_OK);
    }
}
```

In the above example, DSShowHardwareSelectorDlg is used to obtain a valid board designator string for use in an DSAssignBoard call. In this case, the software operator makes an onscreen choice using the hardware selection listbox inside the Hardware Selector. Note that the above example assumes that DSEngineOpen was called at some point to produce a valid engine handle for use in the DSAssignBoard call.

See Also

DSGetHWMgrEntry

DSSetHWMgrEntry

DSReadHWSetupFile

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

Board Designator Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSWriteHWSetupFile

UINT DSWriteHWSetupFile (hHWList)

HGLOBAL hHWList;	// handle to DSP/analog configuration data inside previously opened Hardware Setup File

The DSWriteHWSetupFile function writes current HWLISTENTRY records (containing DSP/analog hardware configuration data) in the global memory area specified by hHWList to its associated Hardware

Setup File (previously specified in calls to DSReadHWSetupFile or DSSetSetupFile).

Parameter	Description
hHWList	handle to global memory area containing DSP/analog hardware configuration data, maintained as series of HWLISTENTRY records

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating a problem in accessing the global memory area or its associated Hardware Setup File. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

Note that if the hHWList parameter is given as NULL, then Hardware Manager determines the correct value assuming the calling application is the current Windows running task.

Currently, the default Hardware Setup File is named hwsetup.lst, and the default location of this file is the location of Hardware Manager (hwmgr.dll or hwmgr32.dll).

Hardware Manager keeps track of different setup files on an application-specific basis. Different applications may use different setup files or the same setup file. Also, different setup files may be open and updated at the same time. The current maximum number of applications which may be using Hardware Manager at one time is eight (8).

It is possible to obtain and change DSP/analog hardware configuration data by calling the DSGetHWMgrEntry and DSSetHWMgrEntry functions with a pointer to a HWLISTENTRY structure. In this case, writing the Hardware Setup File using DSWriteHWSetupFile should be done if DSP/analog hardware configuration data has been changed, and needs to be saved. Alternatively, setup files may be altered directly using a standard text editor, outside of the Hardware Manager environment. Note that if the Hardware Selector is invoked by the operator, and DSP/analog hardware configuration data subsequently

changed followed by pushing the "OK" button, a DSWriteHWSetupFile call is executed automatically.

The hHWList parameter, which may be a return value from a previous call to the DSReadHWSetupFile function is a global memory handle to a series of HWLISTENTRY records containing DSP/analog hardware configuration data. The number of records can be determined by calling DSGetHWMgrNumEntries. The global memory handle is created using the GMEM_SHARED attribute.

NOTE: It is not recommended that calling applications access HWLISTENTRY records directly using the global memory handle returned by DSReadHWSetupFile. Instead, the DSGetHWMgrEntry call should be used.

Note that the current hardware value setting, as well as hardware variable settings, are not stored in the Hardware Setup File. They are instead stored in a separate private configuration file maintained by Hardware Manager (hwmgr.ini), on an application-independent basis (separate sections inside the configuration file are used).

For more information, see Signalogic DSP Software Users Guide section 8.3, Hardware Setup File, section 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure in section 8.3.2.

C/C++ Example

The example below shows the DSWriteHWSetupFile function being called:

```
int nErrorType;

if (!DSWriteHWSetupFile(NULL)) {

    nErrorType = DSGetHWMgrErrorStatus();
    //  do error processing...
}
```

See Also

DSReadHWSetupFile

DSGetHWMgrEntry

DSSetHWMgrEntry

DSGetHWMgrNumEntries

DSShowHardwareSelectorDlg

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in section 8.3.2

2.2.3 Hardware Library Functions

[@k functions,, DSPower-HWLib, functions, Hardware Library, hwlib.dll, hwlib32.dll]

Below are listed functions accessible in the Hardware Library API. For information on how to include Hardware Library (hwlib.dll or hwlib32.dll) in C/C++, Visual Basic, MATLAB, and LabView projects, see Signalogic DSP Software Users Guide section 4.2, Creating Programs for DSPower-HWLib and DSPower-SPLib.

Hardware Library Functions

DSAcquireWvfrmFile

DSAssignBoard

DSCalcSampFreq

DSCancelWaitBuffer

DSCancelWaitFlag

DSDisableBoard

DSDSPToIEEE

DSFreeBoard

DSGenerateWvfrmFile

DSGetBoardBaseAddr

DSGetBoardBusType

DSGetBoardClass

DSGetBoardInfo

DSGetHVarMem

DSGetMem

DSGetMemArch
DSGetMemSize
DSGetWordLength
DSHoldBoard
DSHoldProcessor
DSInitBoard
DSIEEEToDSP
DSLoadFileBoard
DSLoadFileProcessor
DSPutHVarMem
DSPutMem
DSReadBoardReg
DSResetBoard
DSResetProcessor
DSRunBoard
DSRunProcessor
DSSetBoardBaseAddr
DSSetBoardBusType
DSWaitForBuffer
DSWaitForFlag
DSWriteBoardReg

DSAcquireWvfrmFile

UINT DSAcquireWvfrmFile (hBoard,
pConversionInfo)

HBOARD hBoard;	// DSP/analog hardware handle
PCONVERSIONINFO pConversionInfo;	// pointer to CONVERSIONINFO struct containing data acquisition or stimulus & response information

The DSAcquireWvfrmFile function performs a data acquisition or stimulus & response operation, storing data from one or more analog input channels on the specified DSP/analog hardware continuously to waveform file.

Parameter	Description
hBoard	handle to DSP/analog hardware previously obtained from DSAssignBoard call
pConversionInfo	pointer to CONVERSIONINFO structure containing information about the data acquisition or stimulus & response operation to be performed

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating the operation failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

-note about failure: because this is asynchronous function, engine problems (waveform file errors, compile errors, etc.) received via WM_DSPENGINE_PROGSTATUS message

C/C++ Example

```
HLOCAL hConvInfo;
PCONVERSIONINFO pConvInfo;

// create CONVERSIONINFO object memory; use zero-initialization to set most
parameters to default values

hConvInfo = LocalAlloc(LMEM_ZEROINIT | LMEM_MOVEABLE |
LMEM_DISCARDABLE, sizeof(CONVERSIONINFO));

pConvInfo = (PCONVERSIONINFO)LocalLock(hConvInfo);

// file into which to acquire data (including path information)

lstrcpy(pConvInfo->szFilename, szWaveformFilename);

// setup for three (3) channels, init channel list (example is sequential, but could be
// arbitrary channel order)
```

```
pConvInfo->NumChan = 3;          // three channels
for (j=0; j<pConvInfo->NumChan; j++) pConvInfo->ChanList[j] = j;

// setup up sampling rate, amount, gain, trigger information

pConvInfo->Fs =22050;  // 22.05 kHz is desired sampling rate
pConvInfo->NumSamples = 100000L;  // acquire 100,000 samples
lstrcpy(pConvInfo->szGainList, "1");  // see notes in CONVERSIONINFO

// start acquisition

DSAcquireWvfrmFile(hBoard, pConvInfo);

// free and delete hConvInfo memory

LocalUnlock(hConvInfo);
LocalFree(hConvInfo);
```

In the above example, numerous elements of the CONVERSIONINFO structure have been left at their default (or zero) values. For instance, no trigger information has been set, digital offset and scale factors are not active, and the default break mode (store acquired data if user-interrupted early) is in effect.

Note also that the stored data file will actually represent 300,000 total samples, because three channels were specified.

See Also

DSGenerateWvfrmFile

WM_DSPENGINE_PROGSTATUS

DSAssignBoard

DSExecEngineFunction

Related Users Guide Topics

section 4.5, Acquiring Data Using DSPower-HWLib Software

Program Status Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 4.1.1, Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

section 7.3, Executing Functions in DSP Engines

DSAssignBoard

HENGINE DSAssignBoard (hEngine,
pszBoardDesignator,
uBusType,
wIOBaseAddr,
wMemBaseAddr)

```
HENGINE  hEngine;          // engine handle
LPCSTR  pszBoardDesignator; // pointer to board designator string
UINT  uBusType;            // bus type
WORD  wIOBaseAddr;         // I/O base address
WORD  wMemBaseAddr;        // memory base address (if any)
```

The DSAssignBoard function allocates and assigns a handle for the specified DSP/analog hardware type. Some important hardware configuration parameters can be initialized at the same time. The hardware type is specified using a board designator string.

The returned handle can be used in numerous other Hardware Library function calls.

Parameter	Description
hEngine	engine handle previously obtained from DSEngineOpen call
pszBoardDesignator	pointer to null-terminated board designator string
uBusType	bus type of specified DSP/analog hardware as defined by the values below; a NULL value indicates the default Bus Type in the Hardware Setup File should be used

Value	Meaning
DS_AB_PCXT	PC-XT bus type
DS_AB_PCAT	PC-AT bus type
DS_AB_ENABLE80X86INST	enable 80x86 instruction set operations

wIOBaseAddr	I/O base address of specified DSP/analog hardware; a NULL value indicates the default I/O Base Address in the Hardware Setup File should be used
wMemBaseAddr	memory base address, if applicable, of specified DSP/analog hardware; a NULL value indicates the default Memory Base Address in the Hardware Setup File should be used

Returns

The return value is a non-zero DSP/analog hardware handle if successful. Otherwise, the return value is NULL, indicating the DSAssignBoard call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

The pszBoardDesignator parameter is a string value which specifies the DSP/analog hardware type. The Hardware Setup File maintained by Hardware Manager contains the current list of valid board designators. Note that this value may also be conveniently chosen by an operator using the Hardware Selector dialog box. For more information about board designators, see Board Designator Field in Hardware Setup File, in Signalogic DSP Software Users Guide section 8.3.1, Hardware Setup File Fields.

NULL can be used for the uBusType, wIOBaseAddr, and wMemBaseAddr parameters to cause the current Bus Type, I/O Base Address, or Memory Base Address values in the Hardware Setup File to be substituted for these parameters. These values can also be changed at any time using the DSSetBoardBusType and DSSetBoardBaseAddr functions.

Most types of DSP/analog hardware do not require a memory base address. See Related Users Guide Topics below for more information. Also, see Signalogic DSP Software System Guide section 4.3.2, Memory-Mapped Transfer Gotchas.

C/C++ Example

```
hBoard = DSAssignBoard(hEngine, szBoardDesignator, uBusType, uIOBaseAddr,
uMemBaseAddr);
```

```
if (hBoard != NULL) {

    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetEngMgrErrorStatus();
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}

if (!DSInitBoard(hBoard)) {

    // display message to user that board is not installed correctly or not
    // responding
}
```

See Also

DSShowHardwareSelectorDlg

Hardware Selector

Hardware Setup File

DSSetBoardBusType

DSSetBoardBaseAddr

DSEngineOpen

Related Users Guide Topics

section 4.1.1, Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

section 8.2, Hardware Selector

section 8.3, Hardware Setup File

Board Designator Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Bus Type Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

I/O Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Memory Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSCalcSampFreq

DWORD DSCalcSampFreq (hBoard,
SampFreqRequested,
nNumChan,
pnChanList,
pSampFreqActual)

```
HBOARD  hBoard;                  // DSP/analog hardware handle
float  SampFreqRequested;        // desired sampling frequency
short int  nNumChan;             // number of channels
short int far*  pnChanList;      // pointer to channel list
float far*  pSampFreqActual;     // pointer to actual sampling frequency
```

Given a desired sampling frequency, the DSCalcSampFreq function calculates the actual sampling frequency, and corresponding mode or control word value, for the specified DSP/analog hardware. In cases where the hardware does not support, or only partially supports, simultaneous sampling, the number of channels and channel list can also be specified.

The DWORD return value represents a mode or control word which can be sent to the DSP code for use in programming the onboard analog I/O clock generator.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
SampFreqRequested	desired sampling frequency (IEEE 32-bit floating-point value)
nNumChan	number of channels; used only for hardware with analog I/O sampling which is not fully simultaneous
pnChanList	pointer to channel list; must have as many elements as specified by nNumChan; used only for hardware with analog I/O sampling which is not fully simultaneous
pSampFreqActual	pointer to IEEE 32-bit floating-point value

which will contain actual sampling rate upon return

Returns

The return value is a non-zero mode or control word suitable for use by DSP code executing on the DSP/analog hardware to program onboard analog I/O sampling frequency clock generators. Otherwise, the return value is NULL, indicating the DSAssignBoard call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

Most types of DSP/analog hardware are capable of sampling input and output data at a limited range and/or number of discrete rates. The range and number vary tremendously from one type of hardware to another. One common example is hardware with sigma-delta converters offering 16 rates varying from 8 kHz to 48 kHz. Another common example is hardware with successive approximation converters offering 65535 different rates varying from 150 Hz to 200 kHz.

The nNumChan and pnChanList parameters are needed only for DSP/analog hardware with analog I/O sampling which is not fully simultaneous. Otherwise, these parameters should be set to NULL.

Examples where nNumChan and pnChanList are required include hardware with one or more multiplexed analog inputs or outputs and hardware with "channel scan" or "channel group" analog inputs or outputs. In these cases, the DSP Engine uses the nNumChan and pnChanList parameters to calculate a mode or control word reflecting an "aggregate sampling rate", which is some multiple of the requested sampling rate. Note that the pSampFreqActual return value does not reflect this aggregate rate, but always indicates the actual sampling frequency that will be in effect for each channel. pnChanList is not required if the hardware has *only one* multiplexed converter, channel group, etc., in which case the aggregate rate is simply the actual rate multiplied by the number of channels.

The DWORD return value is a mode or control register word which can typically be used by DSP code executing on the DSP/analog hardware, or by the DSP Engine to access a host PC clock-generator

register. In almost all cases, there is usually one register on the DSP/analog hardware which is responsible for sampling frequency clock generation. This may be an onchip timer, an external timer device, a PAL or PEEL, a CODEC control register, etc. Having this value calculated by the DSP Engine provides the main advantage of informing the operator or user of differences between desired and actual sampling rates *a priori*.

When using Hypersignal DSP Source Code Interface software functions or operating modes, the return mode or control word value can be written to the DSP_FSMODE location, and the value pointed to by pSampFreqActual can be written to the DSP_FSVALUE location.

A successful DSInitBoard function must have been completed beforehand.

C/C++ Example

```
DWORD dwFsMode;
float ActualFs, DesiredFs;

DesiredFs  = 75000;  // desired sampling rate is 75 kHz
dwFsMode = DSCalcSampFreq(hBoard, DesiredFs, NULL, NULL, &ActualFs);

DSPutHVarMem(hBoard, DSP_FSMODE, dwFsMode);
DSPutHVarMem(hBoard, DSP_FSVALUE, ActualFs);
```

In the above example, the nNumChan and pnChanList parameters have been ignored; these are only required for some hardware types with non-simultaneous sampling (e.g. inputs which are “scanned”, or sampled on a rotating basis, which repeats).

See Also

DSGetMemSize

DSGetMemArch

DSGetWordLength

DSPutHVarMem

DSP_FSMODE

DSP_FSVALUE

DSGetBoardClass

Related Users Guide Topics

section 4.3.3, Hardware Interrogation
Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4,
Example Code: DSPower-HWLib and DSPower-SPLib Software

DSCancelWaitBuffer

UINT DSCancelWaitBuffer (hBoard,
nBufferNum)

```
HBOARD  hBoard;          // DSP/analog hardware handle
short int  nBufferNum;   // buffer number
```

The DSCancelWaitBuffer function cancels a pending DSP Engine wait-for-buffer request with the specified DSP/analog hardware.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
nBufferNum	number of buffer to cancel wait request for

Returns

The return value TRUE if the wait-for-buffer request was canceled successfully. Otherwise, the return value is NULL, indicating the DSCancelWaitBuffer call failed.

Comments

If a wait-for-buffer request was not pending for the specified buffer number, the DSCancelWaitBuffer call still returns TRUE.

If the wait-for-buffer request was established using interrupt-driven notification (DS_WFB_POLLED for uNotifyType parameter in DSWaitForBuffer), the DSP-to-host PC interrupt is disabled.

C/C++ Example

See Also

DSWaitForBuffer

WM_DSPENGINE_BUFRDY

DSCancelWaitFlag

Related Users Guide Topics

Buffer Ready Messages from DSP Engines, in section 7.4 Registering Callback Messages from DSP Engines

DSCancelWaitFlag

UINT DSCancelWaitFlag (hBoard,
nFlagNum)

```
HBOARD  hBoard;          // DSP/analog hardware handle
short int  nFlagNum;     // flag number
```

The DSCancelWaitFlag function cancels a pending DSP Engine wait-for-semaphore request with the specified DSP/analog hardware.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
nFlagNum	number of flag to cancel wait request

Returns

The return value TRUE if the wait-for-semaphore request was canceled successfully. Otherwise, the return value is NULL, indicating the DSCancelWaitFlag call failed.

Comments

If a wait-for-semaphore request was not pending for the specified flag number, the DSCancelWaitFlag call still returns TRUE.

If the wait-for-semaphore request was established using interrupt-driven notification (DS_WFB_POLLED for uNotifyType parameter in DSWaitForFlag), the DSP-to-host PC interrupt is disabled.

C/C++ Example

See Also

DSWaitForFlag

WM_DSPENGINE_FLGRDY

DSCancelWaitBuffer

Related Users Guide Topics

Semaphore Messages from DSP Engines, in section 7.4 Registering Callback Messages from DSP Engines

DSDisableBoard

UINT DSDisableBoard (hBoard)

HBOARD hBoard; // DSP/analog hardware handle

The DSDisableBoard function disables the specified DSP/analog hardware, placing all DSP processors in reset and disabling other onboard resources which may be active.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call

Returns

The return value is non-zero if the DSP/analog hardware is disabled successfully. Otherwise, the return value is NULL, indicating the DSDisableBoard call failed.

Comments

DSDisableBoard is the antithesis of DSInitBoard. It places all DSP devices on the DSP/analog hardware in reset, and disables other resources that may have been initialized in the course of other Hardware Library calls and/or subsequent DSP code operations. Examples include disabling host PC memory-mapped areas (if any), turning off sampling rate clock generators, disabling output drivers, etc.

After a DSDisableBoard call, a DSResetBoard call should be invoked prior to making further Hardware Library calls to the DSP/analog hardware.

NOTE: DSDisableBoard should not be called unless a successful DSInitBoard function has been completed previously.

C/C++ Example

```
DisableBoard(hBoard);  // shut-down board, including analog I/O
```

See Also

DSInitBoard

DSResetBoard

DSAssignBoard

Related Users Guide Topics

section 4.1.1, Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSDSPToIEEE

UINT DSDSPToIEEE (hBoard,
uIEEEDataSize,
pSrcMem,
pDstMem,
dwCount)

HBOARD hBoard;	// DSP/analog hardware handle
UINT uIEEEDataSize;	// size of each IEEE floating-point data element
void far* pSrcMem;	// pointer to source buffer containing 32-bit floating-point DSP data
void far* pDstMem;	// pointer to destination data buffer which will contain IEEE floating-point data
DWORD dwCount;	// number of data elements to convert

The DSDSPToIEEE function converts DSP 32-bit floating-point data (in format corresponding to DSP devices associated with hBoard) to IEEE 32-bit floating-point data. Both source and destination data buffers are expected to be host PC memory defined by the calling application.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
uIEEEDataSize	size of individual IEEE floating-point data elements, as defined below:

Value	Meaning
DS_DTI_IEEESIZE32	32-bit IEEE data
DS_DTI_IEEESIZE64	64-bit IEEE data

pSrcMem	address in host PC memory of source buffer containing DSP 32-bit floating-point data
pDstMem	address in host PC memory of destination buffer which will contain IEEE 32-bit floating-point data
dwCount	number of data elements to convert

Returns

The return value is a TRUE if the conversion was completed successfully. Otherwise, the return value is NULL, indicating the DSDSPToIEEE call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

The purpose of the DSDSPToIEEE and DSIEEEToDSP functions is to convert calling application floating-point data to and from DSP floating-point data. These functions are most useful before and after DSGetMem and DSPutMem functions.

The DSDSPToIEEE and DSIEEEToDSP functions determine the DSP chip family type from the driver ID value associated with hBoard. This value is established from the Driver ID field in the Hardware Setup File when a DSAssignBoard call is made. Currently, for ***fixed-point DSP devices,*** the DSDSPToIEEE function treats DSP data as if

it were in signed fractional form, which means that only IEEE data values in the range of ±1 will be generated and stored.

C/C++ Example

```
DWORD n,dwCount = 1024;
float dbuf[dwCount];

for n = 0; n<dwCount-1; n++) dbuf[n] = sin(2*pi*n*1000/10000);  // some IEEE data

// convert data to native floating-point DSP format

DSIEEEToDSP(hBoard, DS_DTI_IEEESIZE32, &dbuf, &dbuf, dwCount);

// transfer converted data to memory on DSP board

 DSPutMem(hBoard, DS_GM_LINEAR_DATA, 0x1000,
   DSGetWordLength(hBoard), &dbuf, dwCount);
```

Notes about the above example

1) IEEE data is converted in-place in dbuf, which means trying to use this data again in the host PC program would produce incorrect results

2) The DSP memory address of 0x1000 was chosen arbitrarily; depending on the type of DSP/acquisition board referenced by hBoard, this location could be in onchip SRAM or offchip SRAM

3) The DSGetWordLength function is a convenient method to determine the natural word-size for a given board; in the above example, it is assumed the result will be 32-bits, because of the previous use of the DS_DTI_IEEESIZE32 value

See Also

DSIEEEToDSP

DSGetMem

DSPutMem

Related Users Guide Topics

section 4.3.4, DSP Memory Data Transfers

Example C/C++ Code: Processing Buffer Ready Messages, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSFreeBoard

UINT DSFreeBoard (hBoard)

HBOARD hBoard; // DSP/analog hardware handle

The DSFreeBoard function unallocates and frees the specified board handle.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call

Returns

The return value is non-zero if the DSP/analog hardware handle is freed successfully. Otherwise, the return value is NULL, indicating the DSFreeBoard call failed.

Comments

Further Hardware Library calls using a handle previously freed by DSFreeBoard will return NULL.

DSFreeBoard should be invoked for any active DSP/analog hardware handles before calling applications which are using Hardware Library terminate. It may also be necessary to choose a handle to free if a recent call to DSAssignBoard indicates that all handles are allocated. In this case, the DSAssignBoard result is NULL and the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines return the DSALLBOARDHANDLESALLOCATED error condition.

It is always a good idea to unallocate DSP/analog hardware handles as soon as they are no longer needed. This is because there may be multiple applications using Hardware Manager, requiring more handles at one time than is readily apparent.

C/C++ Example

See Also

DSAssignBoard

Related Users Guide Topics

section 4.1.1, Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSGenerateWvfrmFile

UINT DSGenerateWvfrmFile (hBoard, pConversionInfo)

HBOARD hBoard;	// DSP/analog hardware handle
PCONVERSIONINFO pConversionInfo;	// pointer to CONVERSIONINFO struct containing waveform file playback or signal generation information

The DSGenerateWvfrmFile function performs a waveform playback or continuous signal generation operation, sending data to one or more analog output channels on the specified DSP/analog hardware continuously from waveform file.

Parameter	Description
hBoard	handle to DSP/analog hardware previously obtained from DSAssignBoard call
pConversionInfo	pointer to CONVERSIONINFO structure containing information about the waveform file playback or continuous signal generation operation to be performed

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating the operation failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

-note about failure: because this is asynchronous function, engine problems (waveform file errors, compile errors, etc.) received via WM_DSPENGINE_PROGSTATUS message

C/C++ Example

See Also

DSAcquireWvfrmFile

WM_DSPENGINE_PROGSTATUS

DSAssignBoard

DSExecEngineFunction

Related Users Guide Topics

section 4.5, Acquiring Data Using DSPower-HWLib Software

Program Status Messages from DSP Engines, in section 7.4, Registering Callback Messages from DSP Engines

section 4.1.1, Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

section 7.3, Executing Functions in DSP Engines

DSGetBoardBaseAddr

WORD DSGetBoardBaseAddr (hBoard,
uAddrType)

```
HBOARD hBoard;      // DSP/analog hardware handle
UINT uAddrType;     // type of base address to retrieve
```

The DSGetBoardBaseAddr function retrieves the current I/O or memory base address for the specified DSP/analog hardware.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
uAddrType	type of base address to retrieve; can be one of the

values defined below:

Value	Meaning
DS_GBBA_IOADDR	retrieve I/O base address
DS_GBBA_MEMADDR	retrieve memory base address

Returns

The return value is the current I/O or memory base address for the DSP/analog hardware specified by hBoard. Otherwise, the return value is NULL, indicating the DSGetBoardBaseAddr call failed.

Comments

The values returned by DSGetBoardBaseAddr are either those initialized in the original DSAssignBoard call which created hBoard or set subsequently using a DSSetBoardBaseAddr call.

Note that if either the uIOBaseAddr or uMemBaseAddr parameters in the original DSAssignBoard call were NULL, then these values are initialized to the current I/O Base Address and Memory Base Address values in the Hardware Setup File.

C/C++ Example

See Also

DSSetBoardBaseAddr

DSAssignBoard

Related Users Guide Topics

I/O Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Memory Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSGetBoardBusType

UINT DSGetBoardBusType (hBoard)

```
HBOARD hBoard;                  // DSP/analog hardware handle
```

The DSGetBoardBusType function retrieves the current bus type for the specified DSP/analog hardware.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call

Returns

The return value is the current bus type for the DSP/analog hardware specified by hBoard, shown by the values defined below. Otherwise, the return value is NULL, indicating the DSGetBoardBusType call failed.

Value	Meaning
DS_AB_PCXT	PC-XT bus type
DS_AB_PCAT	PC-AT bus type
DS_AB_ENABLE80X86INST	enable 80x86 instruction set operations

Comments

Depending on the speed of the host PC, and the type of DSP/analog hardware, there may be cases where specifying a different bus type can make the difference between correct operation of the hardware, or improve performance. For example, there are some DSP/analog hardware types which do not operate reliably when 80x86 type "string move" instructions are used, in which case the DS_AB_ENABLE80X86INST bus type attribute shown above should be avoided. For complete bus type information for specific hardware, see chapter 3 in the Hypersignal-Macro Series Hardware Reference Guide.

The value returned by DSGetBoardBusType is either that initialized in the original DSAssignBoard call which created hBoard or set subsequently using a DSSetBoardBusType call.

Note that if the uBusType parameter in the original DSAssignBoard call was NULL, then this values are initialized to the current Bus Type value in the Hardware Setup File.

C/C++ Example

See Also

DSSetBoardBusType

DSAssignBoard

Related Users Guide Topics

Bus Type Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSGetBoardClass

WORD DSGetBoardClass (hBoard)

```
HBOARD hBoard;                  // DSP/analog hardware handle
```

The DSGetBoardClass function retrieves a classification value suitable for use in differentiating between hardware types by DSP source code executing on the DSP/analog hardware specified by hBoard.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call

Returns

The return value is a classification value for the specified DSP/analog hardware. Since classification values may be zero, the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should always be checked after this call (see C/C++ source code example below).

Comments

The DSGetBoardClass function returns a hardware classification value suitable for use by DSP Source Code Interface software. This value is maintained by DSP Engines because DSP Source Code Interface software for one type of DSP chip family may support multiple types of DSP/analog hardware, each with different configuration and analog I/O driver characteristics. Classification values can be sent to executing DSP source code during or prior to run-time initialization.

The result of the DSGetBoardClass function can be sent directly to the Hypersignal DSP Source Code software DSP_BOARDCLASS and DSP_BOARDSUBCLASS variables (see source code example below).

For a list of classification values, see section 3.2, Board Designator, Driver ID, and Classification Table.

A successful DSInitBoard function must have been completed beforehand.

C/C++ Example

See Also

DSCalcSampFreq

DSGetMemSize

DSGetMemArch

DSGetWordLength

DSPutHVarMem

DSP_BOARDCLASS

DSP_BOARDSUBCLASS

Related Users Guide Topics

section 4.3.3, Hardware Interrogation

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSGetBoardInfo

UINT DSGetBoardInfo (hBoard,

uInfoType)

```
HBOARD  hBoard;      // DSP/analog hardware handle
UINT  uInfoType;     // type of information to retrieve
```

The DSGetBoardInfo function retrieves internal Hardware Library information for the specified DSP/analog hardware.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
uInfoType	type of information to retrieve, as indicated by one of the values defined below:

Value	Meaning
DS_GBI_ENGINEHANDLE	engine handle used in original DSAssignBoard call
DS_GBI_DRIVERID	driver ID value
DS_GBI_CALLALLOWED	call groups allowed flag
DS_GBI_HWMGRENTRYINDEX	Hardware Setup File index

Returns

The return value is the specified type of information for the DSP/analog hardware indicated by hBoard. Since all types of return values except for engine handles could legitimately be zero, the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should always be checked after this call (see C/C++ source code example below).

Comments

For more information about driver IDs and the call groups allowed flag, see Signalogic DSP Software Users Guide section 8.3.2, Hardware Setup File Access; see also section 5.1.1 HWLISTENTRY Structure, which describes the structure used to contain individual Hardware Setup File entries.

When uInfoType is DS_GBI_HWMGRENTRYINDEX, the return value can be used directly in the DSGetHWMgrEntry and DSSetHWMgrEntry function calls in Hardware Manager.

C/C++ Example

See Also

DSAssignBoard

DSGetHWMgrEntry

HWLISTENTRY

DSCalcSampFreq

DSGetMemSize

DSGetMemArch

DSGetWordLength

DSGetBoardClass

Related Users Guide Topics

section 4.1.1, Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

HWLISTENTRY Structure, in section 8.3.2, Hardware Setup File Access

section 4.3.3, Hardware Interrogation

DSGetHVarMem

DWORD DSGetHVarMem (hBoard,
dwHVarAddr)

```
HBOARD  hBoard;        // DSP/analog hardware handle
DWORD   dwHVarAddr;    // address of variable
```

The DSGetHVarMem function reads a data element from the specified Hypersignal DSP source code variable in DSP program memory on the DSP/analog hardware indicated by hBoard.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
dwHVarAddr	address of Hypersignal DSP source code variable

to read; for a list of possible variables, see section 3.3, Hypersignal DSP Source Code Variable List

Returns

The return value is the DSP source code variable if read successfully. Since variables can legitimately be zero, the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should always be checked (see C/C++ source code example below).

Comments

The DSGetHVarMem and DSPutHVarMem functions are specific functions designed to transfer data between calling applications and Hypersignal DSP Source Code Interface software which has been loaded or is executing on DSP/analog hardware. Hypersignal DSP Source Code Interface software supports a wide variety of DSP/analog hardware types, and contains several built-in "operating modes" which can be controlled from host PC applications.

For general information, see Signalogic DSP Software Users Guide section 5.0.2, DSP Source Code Interfaces for Hypersignal.

Note that all Hypersignal DSP source code variables are maintained in DSP program memory. Although the dwHVarAddr parameter can be any value, it should be one of the documented Hypersignal DSP source code variables. For a list of Hypersignal DSP source code variables, see section 3.3, Hypersignal DSP Source Code Variable List. In cases where user-defined dwHVarAddr values are used, the actual program memory location accessed will not be these values. The actual address can be calculated as:

```
dwActualAddr = dwHVarAddr – DSP_BOARDCLASS + VarStartAddr
```

DSP_BOARDCLASS is a Hypersignal DSP source code variable, as explained above (it is the first entry in the variable list maintained by Hypersignal DSP Source Code Interface software). In Hypersignal DSP Source Code Interface software, VarStartAddr is specific for each DSP device family. Note also that even though DSGetHVarMem allows for 32-bit values to be read, some DSP devices have a 16-bit or 24-bit natural wordlength. In these cases, only the least significant wordlength bits are read. The DSGetWordLength function can be used to determine wordlength for specified types of DSP/analog hardware. Below is a table with common VarStartAddr and wordlength values.

DSP Family	VarStartAddr	Wordlength
ADSP-21xx	0x1c	16
DSP32C	0x40†	32
DSP3210	0x40†	32
DSP5600x	0x40	24
DSP9600x	0x100	32
TMS320C25	0x42f	16
TMS320C3x	0x400†	32
TMS320C4x	0x400	32
TMS320C5x	0x42f	16

All values for VarStartAddr listed above are given as if DSP device program memory is accessible only as contiguous locations each of wordlength bits. Values marked with † indicate that the DSP device may actually be byte- or word-addressable, but only the wordlength value shown is used for DSGetHVarMem and DSPutHVarMem addressing.

C/C++ Example

Below is an example of reading a DSP source code variable:

```
DWORD dwTimDataBaseAddr;

dwTimDataBaseAddr = DSGetHVarMem(hBoard, DSP_TIMDATAADDR);

if (!dwTimDataBaseAddr) {

    HENGINE hEngine = DSGetBoardInfo(hBoard, DS_GBI_ENGINEHANDLE);
    int nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();

    if (nErrorType != NULL) {

        // do error processing...
    }
}
```

The example above reads the default time data base address used by native Hypersignal DSP Source Code Interface functions and instruments. This example assumes a DSLoadFileBoard or DSLoadFileProcessor calls has previously been invoked, which would

cause Hypersignal DSP source code variables to be initialized, depending on the program file that was downloaded.

See Also

DSPutHVarMem

DSGetMem

DSPutMem

DSGetWordLength

DSDSPToIEEE

DSIEEEToDSP

Related Users Guide Topics

section 4.3.4, DSP Memory Data Transfers

section 5.0.1, DSP Source Code Interface Documentation

section 5.0.2, DSP Source Code Interfaces for Hypersignal

section 5.0.3, DSP Source Code Interfaces for Hypersignal in Engine Mode

DSGetMem

UINT DSGetMem (hBoard,
uMemType,
dwDSPMemAddr,
uBitWidth,
pMem,
dwCount)

```
HBOARD  hBoard;          // DSP/analog hardware handle
UINT  uMemType;          // DSP memory type
DWORD  dwDSPMemAddr;     // DSP memory start address
UINT  uBitWidth;         // data element width in DSP memory
void far*  pMem;         // pointer to buffer to receive data
DWORD  dwCount;          // number of data elements to transfer
```

The DSGetMem function transfers one or more data elements from DSP/analog hardware memory to host PC memory defined by the calling application.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
uMemType	type of DSP memory from which to transfer data elements, as defined below:

Value	Meaning
DS_GM_LINEAR_PROGRAM	linear program memory
DS_GM_LINEAR_DATA	linear data memory
DS_GM_LINEAR_DATA_RT	linear data memory fast transfer while DSP is in run state (valid for some DSPs only)
DS_GM_VECTOR_DATA_X	vector X: data memory
DS_GM_VECTOR_DATA_Y	vector Y: data memory

dwDSPMemAddr	starting address in DSP data memory for transfer
uBitWidth	data element width in DSP memory for transfers, as defined below:

Value	Meaning
DS_GM_SIZE8	DSP memory data element width is 1 byte
DS_GM_SIZE16	DSP memory data element width is WORD
DS_GM_SIZE24	DSP memory data element width is 3 bytes
DS_GM_SIZE32	DSP memory data element width is DWORD or float

pMem	pointer to calling application memory which will receive data elements
dwCount	number of data elements to transfer

Returns

The return value is a TRUE if the transfer was completed successfully. Otherwise, the return value is NULL, indicating the DSGetMem call failed. For further error checking, use the DSGetEngineErrorStatus

and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

The DSGetMem and DSPutMem functions are flexible, general-purpose functions intended to transfer data between host PC memory and onboard or onchip DSP/analog hardware memory. Because of the wide variety of DSP/analog hardware memory types, there are several items to keep in mind:

1) Not all DSP devices have onchip memory which is accessible from outside the chip, and of those that do, not all board designs "project" accessibility to the host PC interface. In these cases, using the DSGetMem and DSPutMem functions with onchip DSP memory addresses will produce undefined results. Consult DSP device and DSP/analog hardware documentation.

2) For almost all types of DSP/analog hardware, not all data element sizes of uBitWidth above are supported. In most cases, DS_GM_SIZE8 is not supported. For all hardware types except those based on Motorola DSP5600x and NEC77220 the DS_GM_SIZE24 value is not supported. In cases where a uBitWidth value is not supported, the transfer defaults to the next higher data element size.

3) Currently, no dynamic data conversion between integer and floating-point or IEEE floating-point and native DSP floating-point formats is made by DSGetMem or DSPutMem. This may change in later versions of Hardware Library, as new uMemType options are added. Thus, for example, if a calling MATLAB application is transferring 64-bit floating-point data to a DSP program, the DSP program may need to "know" to convert this data to its native 32-bit floating-point format before processing. Note, however, that for 32-bit floating-point DSP devices which do not use IEEE format (which is the default format used in Borland, Microsoft, and WATCOM C/C++ compilers), the DSDSPToIEEE and DSIEEEToDSP routines are available for converting native DSP 32-bit floating-point data to and from IEEE 32-bit floating-point format. Note also that these routines only convert data using calling application memory buffers, and do not involve DSP memory.

4) For 32-bit floating-point DSP devices which support byte- or word-addressable memory, addressing used by DSGetMem and DSPutMem appears as if DSP memory is accessible as contiguous locations each of

uBitWidth bits. For example, if a DSP device supports byte-addressable memory, then using a uBitWidth value of DS_GM_SIZE16 would cause access to occur at the *byte address* of dwDSPDataAddr∗2, and a uBitWidth value of DS_GM_SIZE32 would cause access to occur at the byte address of dwDSPMemAddr∗4 (or the *word address* of dwDSPMemAddr∗2). Examples of byte-addressable DSP devices include the AT&T DSP32C and DSP3210, and Texas Instruments TMS320C32.

5) The DS_GM_LINEAR_DATA_RT uMemType is intended for transferring 16-bit analog I/O data to DSP programs as fast as possible while the DSP device is in the run state. Currently, using the DS_GM_LINEAR_DATA_RT attribute will make a difference only with Texas Instruments TMS320 series devices. With all other devices, results are the same as using DS_GM_LINEAR_DATA. This attribute is provided because it turns out that data transfer for many TMS320 series based DSP/analog hardware designs is faster if only the high 16-bits of data memory are used when transferring data to DSP memory, or judicious use (or non-use) of the hold state built in to these devices is required. In all cases except for TMS320C25 devices, and situations where a Hypersignal DSP Source Code Interface operating mode is being used which expects high-word transfers, it is safest to use the DS_GM_LINEAR_DATA construct.

6) Currently, the DS_GM_VECTOR_DATA_X and DS_GM_VECTOR_DATA_Y uMemType attributes are only supported for Motorola DSP5600x and DSP9600x devices. Regardless of the DSP type, DS_GM_VECTOR_DATA_X will produce the same result as DS_GM_LINEAR_DATA.

7) Experience has shown that it would be difficult to specify a straightforward processor list parameter in DSP memory data transfer functions. For multiprocessor DSP/analog hardware designs, data memory is generally shared, and different processor data spaces are chosen using memory address values, which allows DSGetMem and DSPutMem to be used as is. In cases where data or program memory between processors is physically separate, DSWriteBoardReg and DSReadBoardReg calls may need to be made to select a memory space before using data transfer functions.

C/C++ Example

The source code example below transfers 32-bit data from DSP data memory address 0xc0000 to a calling application data buffer created by the Windows GlobalAlloc function. Note that the actual type of data being transferred could be either 32-bit integer or 32-bit floating-point, although the example assumes the data is floating-point.

```
HGLOBAL hBuf;
float far* pBuf;
WORD wBuflen;
DWORD dwDSPBaseAddr;

dwDSPBaseAddr = 0xc0000;

hBuf = GlobalAlloc(GMEM_MOVEABLE, wBufLen*sizeof(float));
pBuf = (float far*)GlobalLock(hBuf);

UINT uStatus = DSGetMem(hBoard, DS_GM_LINEAR_DATA, dwDSPBaseAddr,
DS_GM_SIZE32, pBuf, wBuflen);

if (!uStatus) {

    HENGINE hEngine = DSGetBoardInfo(hBoard, DS_GBI_ENGINEHANDLE));
    int nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    itoa(nErrorType, tmpstr, 10);
    MessageBox(NULL, tmpstr, "DSGetMem error occurred", MB_OK);
}
GlobalUnlock(hBuf);  // release mem
```

Note in the above example that a 32-bit floating-point device is assumed. The above example would not be recommended for a 16-bit or 24-bit fixed-point DSP device.

See Also

DSPutMem

DSGetHVarMem

DSPutHVarMem

DSWaitForBuffer

WM_DSPENGINE_BUFRDY

DSGetWordLength

DSDSPToIEEE

DSIEEEToDSP

Related Users Guide Topics

section 4.3.4, DSP Memory Data Transfers

Example C/C++ Code: Processing Buffer Ready Messages, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSGetMemArch

UINT DSGetMemArch (hBoard)

```
HBOARD hBoard;          // DSP/analog hardware handle
```

The DSGetMemArch function retrieves the memory architecture for the DSP/analog hardware specified by hBoard.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call

Returns

If the DSGetMemArch function is successful, the return value is the type of memory architecture used on the specified DSP/analog hardware, as defined by the values below. Otherwise, the return value is NULL and the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should be checked for error conditions.

Value	Meaning
DS_GMA_LINEAR	memory architecture is linear, with program and data memory in the same address space
DS_GMA_HARVARD	memory architecture is modified Harvard, with program and data memory in separate address spaces
DS_GMA_VECTOR	memory architecture is modified Harvard, except data memory is further subdivided into two (2) vector spaces

Comments

The DSGetMemArch function returns a memory architecture type value suitable for use in DSGetMem and DSPutMem functions (see source code example below).

Typical examples of DSP devices which have a linear memory architecture are AT&T DSP32C and DSP3210, and Texas Instruments TMS320C3x and TMS320C4x. Typical examples of devices which have a modified Harvard memory architecture are Texas Instruments TMS320C5x. Typical examples of devices which have a vector memory architecture are Motorola DSP5600x.

A successful DSInitBoard function must have been completed beforehand.

C/C++ Example

See Also

DSPutMem

DSGetMem

DSCalcSampFreq

DSGetMemSize

DSGetWordLength

DSGetBoardClass

Related Users Guide Topics

section 4.3.3, Hardware Interrogation

Example C/C++ Code: Processing Buffer Ready Messages, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSGetMemSize

DWORD DSGetMemSize (hBoard)

HBOARD hBoard;	// DSP/analog hardware handle
WORD wProcessorNum;	// value specifying which processor's data memory to size

The DSGetMemSize function retrieves the maximum data memory size for the processor specified by wProcessorNum on the DSP/analog hardware specified by hBoard.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
wProcessorNum	value containing processor list; each bit represents a processor, starting with bit 0 for processor 0; only 1 bit should be set

Returns

If the DSGetMemSize function is successful, the return value is the maximum data memory size available for use with the specified processor. Otherwise, the return value is NULL and the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should be checked for error conditions.

Comments

The DSGetMemSize function will return NULL in the following cases:

1. the wProcessorNum parameter indicates more than processor 0, and the DSP/analog hardware has only one processor (or in general indicates a higher processor number than actually exists)
2. the wProcessorNum parameter is zero
3. the wProcessorNum parameter has more than one bit set

The DSGetMemSize function returns the maximum amount of data memory detected for the processor specified by wProcessorNum. For most hardware types, the return value is in units of the native wordlength of the DSP device. Some examples include 32-bit words for TMS320C3x and TMS320C4x devices, 24-bit words for DSP5600x devices, and 16-bit words for ADSP-21xx and TMS320C5x devices. However, there are currently two (2) exceptions. The DSGetMemSize return value is in 16-bit words for DSP32C and DSP3210 devices, even though these devices are 32-bit floating-point. This is because these devices have true multiwidth access to onchip and external memory, and some DSP Engine drivers take advantage of this feature

to store data (especially analog I/O data) in compact (16-bit integer) form when possible.

In cases where the DSP/analog hardware has separate program and data memory areas, the DSGetMemSize function does not measure program memory.

In some cases, the return value may either be somewhat less than or exceed the amount of installed onboard, external data memory. This may be because external memory addressing is contiguous with onchip memory, or overlaps onchip memory (creating an unusable region).

A successful DSInitBoard function must have been completed beforehand.

C/C++ Example

See Also

DSGetMemArch

DSGetWordLength

DSPutMem

DSGetMem

DSCalcSampFreq

DSGetBoardClass

Related Users Guide Topics

section 4.3.3, Hardware Interrogation

DSGetWordLength

WORD DSGetWordLength (hBoard)

```
HBOARD hBoard;                  // DSP/analog hardware handle
```

The DSGetWordLength function retrieves the natural word length of the DSP device(s) on the DSP/analog hardware specified by hBoard.

Parameter	Description

hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call

Returns

If the DSGetWordLength function is successful, the return value is the natural wordlength, as defined in the table below, of the DSP device(s) used on the DSP/analog hardware. Otherwise, the return value is NULL and the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should be checked for error conditions.

Value	Meaning
DS_GM_SIZE16	DSP device is 16-bit fixed-point
DS_GM_SIZE24	DSP device is 24-bit fixed-point
DS_GM_SIZE32	DSP device is 32-bit floating-point

Comments

Some typical examples of DSGetWordLength return values:

AT&T DSP32C, DSP3210	DS_GMA_SIZE32
Analog Devices ADSP-21xx	DS_GMA_SIZE16
Analog Devices ADSP-21xxx	DS_GMA_SIZE32
Motorola DSP5600x	DS_GMA_SIZE24
Motorola DSP9600x	DS_GMA_SIZE32
Texas Instruments TMS320C25, TMS320C5x	DS_GMA_SIZE16
Texas Instruments TMS320C3x, TMS320C4x	DS_GMA_SIZE32

Note that this function does not actually "touch" hardware. The Driver ID field in the Hardware Setup File is used to determine the DSP chip family.

C/C++ Example

```
wDSPWordLength = DSGetWordLength(hBoard);

if (wDSPWordLength == NULL) {
```

```
    hEngine = DSGetBoardInfo(hBoard, DS_GBI_ENGINEHANDLE);
    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    // do error processing....
}
```

See Also

DSGetMemSize

DSGetMemArch

DSPutMem

DSGetMem

DSCalcSampFreq

DSGetBoardClass

Related Users Guide Topics

section 4.3.3,Hardware Interrogation

DSHoldBoard

UINT DSHoldBoard (hBoard)

```
HBOARD hBoard;                  // DSP/analog hardware handle
```

The DSHoldBoard function places all DSP processors on the DSP/analog hardware specified by hBoard into the hold state, if the hardware and DSP devices support this state.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call

Returns

The return value is non-zero if all DSP processors were placed in the hold state successfully, or if the DSP/analog hardware and/or DSP devices do not support this mode. Otherwise, the return value is NULL, indicating the DSHoldBoard call failed.

Comments

Upon successful return, all DSP devices on the DSP/analog hardware will be in the hold state--as long as a hold state is supported by the DSP/analog hardware host PC interface and the DSP device itself. For example, most Texas Instruments TMS320 series devices can be placed in hold state; only some of the DSP/analog hardware types with these processors allow hold state to be enabled from the host PC. Other types of DSP devices do not offer an externally-selectable hold state.

All DSP devices are held simultaneously, if the DSP/analog hardware supports a simultaneous-write mode. For multiprocessor hardware, if only certain DSP processors need to be held, use the DSHoldProcessor function.

Note that it varies between hardware types whether data transfer operations such as DSGetMem and DSPutMem may be executed while DSP devices are in the hold state. See documentation for the DSP/analog hardware type for specific information.

C/C++ Example

See Also

DSHoldProcessor

DSResetBoard

DSResetProcessor

DSRunBoard

DSRunProcessor

Related Users Guide Topics

section 4.3.1, Hardware and Processor Control

DSHoldProcessor

UINT DSHoldProcessor (hBoard,
wProcessorList)

HBOARD hBoard; // DSP/analog hardware handle

```
WORD  wProcessorList;          // value specifying which processors should
                               be held
```

The DSHoldProcessor function places one or more processors on the DSP/analog hardware specified by hBoard into the hold state, if the hardware and DSP devices support this state. The wProcessorList parameter is used to indicate which processors should be reset.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
wProcessorList	value containing processor list; each bit represents a processor, starting with bit 0 for processor 0; a 1 causes the processor to be included, a 0 causes it to be unaffected

Returns

The return value is non-zero if the DSP processors specified by wProcessorList were held successfully, or if the DSP/analog hardware and/or DSP devices do not support this mode. Otherwise, the return value is NULL, indicating the DSHoldProcessor call failed.

Comments

The DSHoldProcessor function will return NULL in the following cases:

1. the wProcessorList parameter indicates more than processor 0, and the DSP/analog hardware has only one processor
2. the wProcessorList parameter is zero
3. the wProcessorList parameter has more bits set to one than there are processors on the DSP/analog hardware

Upon successful return, DSP devices on the DSP/analog hardware specified by wProcessorList will be in the hold state--as long as a hold state is supported by the DSP/analog hardware host PC interface and the DSP devices themselves. For example, most Texas Instruments TMS320 series devices can be placed in hold state; only some of the DSP/analog hardware types with these processors allow hold state to be enabled from the host PC. Other types of DSP devices do not offer an externally-selectable hold state.

DSP devices specified by wProcessorList are held simultaneously, if the DSP/analog hardware supports a simultaneous-write mode. For multiprocessor hardware, if all DSP processors need to be held simultaneously, it is simpler to use the DSHoldBoard function.

Note that it varies between hardware types whether data transfer operations such as DSGetMem and DSPutMem may be executed while DSP devices are in the hold state. See documentation for the DSP/analog hardware type for specific information.

C/C++ Example

See Also

DSHoldBoard

DSRunProcessor

DSRunBoard

DSResetBoard

DSResetProcessor

Related Users Guide Topics

section 4.3.1,Hardware and Processor Control

DSInitBoard

UINT DSInitBoard (hBoard)

HBOARD hBoard; // DSP/analog hardware handle

The DSInitBoard function checks for the presence, and if found, initializes the specified DSP/analog hardware.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call

Returns

The return value is non-zero if the DSP/analog hardware is located and initialized successfully. Otherwise, the return value is NULL, indicating the DSInitBoard call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

Initialization of DSP/analog hardware is straightforward, and occurs in the following steps:

1. A short register or onboard memory check is made to verify that the hardware is installed correctly.
2. Any DSP devices on the hardware are placed in reset state.
3. For those types of hardware which require it, a resident "host interface" DSP code segment may be downloaded and allowed to run. This is necessary in cases where the design of the hardware and/or DSP device does not allow onboard memory to be accessed while the DSP device is held in reset.

Note that in all cases, if the DSInitBoard function is successful, further operations such as DSGetMem and DSPutMem can be executed at any time. *If not, then further operations will produce undefined results*. The one exception to this is attempting additional DSInitBoard calls after using DSSetBoardBaseAddr to alter the I/O Base Address and/or Memory Base Address values for the hardware.

C/C++ Example

See Also

DSAssignBoard

DSLoadFileBoard

Related Users Guide Topics

section 4.1.1, Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSIEEEToDSP

UINT DSIEEEToDSP (hBoard,
uIEEEDataSize,
pSrcMem,
pDstMem,
dwCount)

```
HBOARD  hBoard;          // DSP/analog hardware handle
UINT  uIEEEDataSize;     // size of each IEEE floating-point data element
void far*  pSrcMem;      // pointer to source buffer containing IEEE floating-point data
void far*  pDstMem;      // pointer to destination buffer which will contain 32-bit floating-point DSP data
DWORD  dwCount;          // number of data elements to convert
```

The DSIEEEToDSP function converts IEEE 32-bit floating-point data to DSP 32-bit floating-point data (in format corresponding to DSP devices associated with hBoard). Both source and destination data buffers are expected to be host PC memory defined by the calling application.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
uIEEEDataSize	individual size of IEEE floating-point data elements, as defined by values below:

Value	Meaning
DS_DTI_IEEESIZE32	32-bit IEEE data
DS_DTI_IEEESIZE64	64-bit IEEE data

pSrcMem	address in host PC memory of source buffer containing IEEE floating-point data
pDstMem	address in host PC memory of destination buffer which will contain DSP 32-bit floating-point data
dwCount	number of data elements to convert

Returns

The return value is a TRUE if the conversion was completed successfully. Otherwise, the return value is NULL, indicating the DSIEEEToDSP call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

The purpose of the DSDSPToIEEE and DSIEEEToDSP functions is to convert calling application floating-point data to and from DSP floating-point data. These functions are most useful before and after DSGetMem and DSPutMem functions.

The DSDSPToIEEE and DSIEEEToDSP functions determine the DSP chip family type from the driver ID value associated with hBoard. This value is established from the Driver ID field in the Hardware Setup File when a DSAssignBoard call is made. Currently, for ***fixed-point DSP devices,*** the DSIEEEToDSP function attempts to convert IEEE data to signed fractional notation, which means that IEEE data values outside of the range ±1 will "wrap around".

C/C++ Example

```
DWORD n,dwCount = 1024;
float dbuf[dwCount];

// get native floating-point DSP data from memory on DSP board

 DSGetMem(hBoard, DS_GM_LINEAR_DATA, 0x1000,
   DSGetWordLength(hBoard), &dbuf, dwCount);

// convert data to IEEE floating-point format

DSDSPToIEEE(hBoard, DS_DTI_IEEESIZE32, &dbuf, &dbuf, dwCount);
```

Notes about the above example

1) native DSP format data is converted in-place in dbuf, which allows the data to be used subsequently in the host PC program by referencing dbuf; attempting to perform floating-point operations on dbuf data after the DSGetMem function but before the DSDSPToIEEE function would produce incorrect results

2) The DSP memory address of 0x1000 was chosen arbitrarily; depending on the type of DSP/acquisition board referenced by hBoard, this location could be in onchip SRAM or offchip SRAM

3) The DSGetWordLength function is a convenient method to determine the natural word-size for a given board; in the above example, it is assumed the result will be 32-bits, because of the DS_DTI_IEEESIZE32 value used in the conversion

See Also

DSDSPToIEEE

DSGetMem

DSPutMem

Related Users Guide Topics

section 4.3.4, DSP Memory Data Transfers

Example C/C++ Code: Processing Buffer Ready Messages, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSLoadFileBoard

UINT DSLoadFileBoard (hBoard,
pszCodeFile)

```
HBOARD  hBoard;          // DSP/analog hardware handle
LPCSTR  pszCodeFile;     // pointer to executable DSP code file string
```

The DSLoadFileBoard function downloads an executable DSP code file to DSP program memory on the specified DSP/analog hardware. If the hardware contains more than one DSP processor with separate program memory, all processors receive the same executable code file.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
pszCodeFile	pointer to null-terminated string containing pathname of

the executable DSP code file to download

Returns

The return value is TRUE if the executable DSP code file is downloaded successfully. Otherwise, the return value is NULL, indicating the DSLoadFileBoard call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

Successful DSAssignBoard and DSInitBoard calls must have been completed before DSLoadFileBoard calls are attempted.

If the pszCodeFile parameter is NULL, the DSLoadFileBoard call will download the current DSP Program File entry in the Hardware Setup File for the corresponding DSP/analog hardware type. This value can be retrieved independently by a call to DSGetHWMgrEntry using a Hardware Setup File index. There are two ways to get this index:

1. call DSGetHWMgrEntryIndex with the original board designator string used in DSAssignBoard
2. call DSGetBoardInfo with the hBoard parameter (same as above) and DS_GBI_HWMGRENTRYINDEX as input

Using the resulting index in a DSGetHWMgrEntry call will then retrieve a HWLISTENTRY structure, in which the pszCodeFile element is the current executable code file.

DSP executable code files should be in the format produced by the DSP chip manufacturer's linker utility. Typically, but not in all cases, DSP chip linkers produce COFF (common object file format) files.

NOTE: All DSP devices on the DSP/analog hardware should be in reset state before the DSLoadFileBoard call is invoked. Reset state is in effect after DSInitBoard, DSResetBoard, DSResetProcessor, and DSDisableBoard calls. Otherwise, results are undefined.

C/C++ Example

```
// load specified DSP program (code) file (if board is multiple-processor, then all
// processors are loaded with same file; see DSLoadFileProcessor below for
// processor-specific program file download)
```

```
if (!DSLoadFileBoard(hBoard, “dspcode.out”)) {

    // try default code file instead

    if (!DSLoadFileBoard(hBoard, NULL)) {

        MessageBox(NULL, " ", "Problem loading DSP program file", MB_OK);
    }
}
```

See Also

DSLoadFileProcessor

DSAssignBoard

DSInitBoard

DSP Program File

Related Users Guide Topics

section 4.1.1, Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSP Program File Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSLoadFileProcessor

UINT DSLoadFileProcessor (hBoard,
pszCodeFile,
wProcessorList)

HBOARD hBoard;	// DSP/analog hardware handle
LPCSTR pszCodeFile;	// pointer to executable DSP code file string
WORD wProcessorList;	// value specifying which processors should receive the DSP code file

The DSLoadFileProcessor function downloads an executable DSP code file to DSP program memory for one or more processors on the specified DSP/analog hardware. The wProcessorList parameter is

used to indicate which processors should receive the executable DSP code file.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
pszCodeFile	pointer to null-terminated string containing pathname of the executable DSP code file to download
wProcessorList	value containing processor list; each bit represents a processor, starting with bit 0 for processor 0; a 1 enables downloading, a 0 disables

Returns

The return value is TRUE if the executable DSP code file is downloaded successfully. Otherwise, the return value is NULL, indicating the DSLoadFileProcessor call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

Successful DSAssignBoard and DSInitBoard calls must have been completed before DSLoadFileProcessor calls are attempted.

The DSLoadFileProcessor function will return NULL in the following cases:

1. the wProcessorList parameter indicates more than processor 0, and the DSP/analog hardware has only one processor
2. the wProcessorList parameter is zero
3. the wProcessorList parameter has more bits set to one than there are processors on the DSP/analog hardware

If the pszCodeFile parameter is NULL, the DSLoadFileProcessor call will download the current DSP Program File entry in the Hardware Setup File for the corresponding DSP/analog hardware type. This value can be retrieved independently by a call to DSGetHWMgrEntry using a Hardware Setup File index. There are two ways to get this index:

1. call DSGetHWMgrEntryIndex with the original board designator string used in DSAssignBoard
2. call DSGetBoardInfo with the hBoard parameter (same as above) and DS_GBI_HWMGRENTRYINDEX as input

Using the resulting index in a DSGetHWMgrEntry call will then retrieve a HWLISTENTRY structure, in which the pszCodeFile element is the current executable code file.

DSP executable code files should be in the format produced by the DSP chip manufacturer's linker utility. Typically, but not in all cases, DSP chip linkers produce COFF (common object file format) files.

NOTE: All DSP devices on the DSP/analog hardware should be in reset state before the DSLoadFileBoard call is invoked. Reset state is in effect after DSInitBoard, DSResetBoard, DSResetProcessor, and DSDisableBoard calls. Otherwise, results are undefined.

C/C++ Example

```
// load specified DSP program (code) file (processor 0 only)

if (!DSLoadFileProcessor(hBoard, “dspcode.out”, 0x01)) {

    // try default code file (processors 0, 1, and 2)

    if (!DSLoadFileProcessor(hBoard, NULL, 0x07)) {

        MessageBox(NULL, " ", "Problem loading DSP program file", MB_OK);
    }
}
```

See Also

DSLoadFileBoard

DSAssignBoard

DSInitBoard

DSP Program File

Related Users Guide Topics

section 4.1.1, Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSP Program File Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSPutHVarMem

UINT DSPutHVarMem (hBoard,
dwHVarAddr,
dwHVarData)

```
HBOARD  hBoard;        // DSP/analog hardware handle
DWORD   dwHVarAddr;    // address of variable
DWORD   dwHVarData;    // data to send
```

The DSPutHVarMem function writes a data element to the specified Hypersignal DSP source code variable in DSP program memory on the DSP/analog hardware indicated by hBoard.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
dwHVarAddr	address of Hypersignal DSP source code variable to be written; for a list of possible variables, see section 3.3, Hypersignal DSP Source Code Variable List
dwHVarData	data to write to Hypersignal DSP source code variable

Returns

The return value is a TRUE if the variable was written successfully. Otherwise, the return value is NULL, indicating the DSPutHVarMem call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

The DSGetHVarMem and DSPutHVarMem functions are specific functions designed to transfer data between calling applications and

Hypersignal DSP Source Code Interface software which has been loaded or is executing on DSP/analog hardware. Hypersignal DSP Source Code Interface software supports a wide variety of DSP/analog hardware types, and contains several built-in "operating modes" which can be controlled from host PC applications.

For general information, see Signalogic DSP Software Users Guide section 5.0.2, DSP Source Code Interfaces for Hypersignal.

Note that all Hypersignal DSP source code variables are maintained in DSP program memory. Although the dwHVarAddr parameter can be any value, it should be one of the documented Hypersignal DSP source code variables. For a list of Hypersignal DSP source code variables, see section 3.3, Hypersignal DSP Source Code Variable List. In cases where user-defined dwHVarAddr values are used, the actual program memory location accessed will not be these values. The actual address can be calculated as:

```
dwActualAddr = dwHVarAddr – DSP_BOARDCLASS + VarStartAddr
```

DSP_BOARDCLASS is a Hypersignal DSP source code variable, as explained above (it is the first entry in the variable list maintained by Hypersignal DSP Source Code Interface software). In Hypersignal DSP Source Code Interface software, VarStartAddr is specific for each DSP device family. Note also that even though dwHVarData allows for 32-bit values to be written, some DSP devices have a 16-bit or 24-bit natural wordlength. In these cases, only the least significant wordlength bits are written. The DSGetWordLength function can be used to determine wordlength for specified types of DSP/analog hardware. Below is a table with common VarStartAddr and wordlength values.

DSP Family	VarStartAddr	Wordlength
ADSP-21xx	0x1c	16
DSP32C	0x40†	32
DSP3210	0x40†	32
DSP5600x	0x40	24
DSP9600x	0x100	32
TMS320C25	0x42f	16
TMS320C3x	0x400†	32
TMS320C4x	0x400	32

TMS320C5x	0x42f	16

All values for VarStartAddr listed above are given as if DSP device program memory is accessible only as contiguous locations each of wordlength bits. Values marked with † indicate that the DSP device may actually be byte- or word-addressable, but only the wordlength value shown is used for DSGetHVarMem and DSPutHVarMem addressing.

C/C++ Example

Below is a short excerpt from the dscope.cpp source code example file included with DSPower-HWLib software:

```
DWORD dwFsMode;
WORD wBoardClass;
float FsActual;

// interrogate engine for board type value
wBoardClass = DSGetBoardClass(hBoard);

// find samp rate ctrl. reg. value and actual rate (closest rate possible to desired)
// 22.05 kHz sampling rate, assume simultaneous sampling

dwFsMode = DSCalcSampFreq(hBoard, 22050.0, NULL, NULL, &FsActual);

// load DSP Program File (usually a COFF file produced by DSP chip linker)
if (!DSLoadFileProcessor(hBoard, NULL, 0x01)) {  // load default file (proc 0 only)

    MessageBox(NULL, " ", "Dscope Demo:  problem loading file", MB_OK);
    goto cleanup;
}

// send down Hypersignal DSP source code variables in anticipation of using a DSP
source code operating mode
...
...
DSPutHVarMem(hBoard, DSP_BOARDCLASS, wBoardClass & 0x00ff);
DSPutHVarMem(hBoard, DSP_BOARDSUBCLASS, wBoardClass >> 8);
...
DSPutHVarMem(hBoard, DSP_FSMODE, dwFsMode);
...
```

See Also

DSGetHVarMem

DSGetMem

DSPutMem

DSGetWordLength

DSDSPToIEEE

DSIEEEToDSP

Related Users Guide Topics

section 4.3.4, DSP Memory Data Transfers

section 5.0.1, DSP Source Code Interface Documentation

section 5.0.2, DSP Source Code Interfaces for Hypersignal

section 5.0.3, DSP Source Code Interfaces for Hypersignal in Engine Mode

DSPutMem

UINT DSPutMem (hBoard,
uMemType,
dwDSPMemAddr,
uBitWidth,
pMem,
dwCount)

```
HBOARD  hBoard;          // DSP/analog hardware handle
UINT  uMemType;          // DSP memory type
DWORD  dwDSPMemAddr;     // DSP memory start address
UINT  uBitWidth;         // data element width in DSP memory
void far*  pMem;         // pointer to buffer to receive data
DWORD  dwCount;          // number of data elements to transfer
```

The DSPutMem function transfers one or more data elements from host PC memory defined by the calling application to DSP/analog hardware memory.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call

uMemType — type of DSP memory to which to transfer data elements, as defined below:

Value	Meaning
DS_GM_LINEAR_PROGRAM	linear program memory
DS_GM_LINEAR_DATA	linear data memory
DS_GM_LINEAR_DATA_RT	linear data memory fast transfer while DSP is in run state (valid for some DSPs only)
DS_GM_VECTOR_DATA_X	vector X: data memory
DS_GM_VECTOR_DATA_Y	vector Y: data memory

dwDSPMemAddr — starting address in DSP data memory for transfer

uBitWidth — data element width in DSP memory for transfers, as defined below:

Value	Meaning
DS_GM_SIZE8	DSP memory data element width is 1 byte
DS_GM_SIZE16	DSP memory data element width is WORD
DS_GM_SIZE24	DSP memory data element width is 3 bytes
DS_GM_SIZE32	DSP memory data element width is DWORD or float

pMem — pointer to calling application memory which contains data elements to transfer

dwCount — number of data elements to transfer

Returns

The return value is a TRUE if the transfer was completed successfully. Otherwise, the return value is NULL, indicating the DSPutMem call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

The DSGetMem and DSPutMem functions are flexible, general-purpose functions intended to transfer data between host PC memory and onboard or onchip DSP/analog hardware memory. Because of the

wide variety of DSP/analog hardware memory types, there are several items to keep in mind:

1) Not all DSP devices have onchip memory which is accessible from outside the chip, and of those that do, not all board designs "project" accessibility to the host PC interface. In these cases, using the DSGetMem and DSPutMem functions with onchip DSP memory addresses will produce undefined results. Consult DSP device and DSP/analog hardware documentation.

2) For almost all types of DSP/analog hardware, not all data element sizes of uBitWidth above are supported. In most cases, DS_GM_SIZE8 is not supported. For all hardware types except those based on Motorola DSP5600x and NEC77220 the DS_GM_SIZE24 value is not supported. In cases where a uBitWidth value is not supported, the transfer defaults to the next higher data element size.

3) Currently, no dynamic data conversion between integer and floating-point or IEEE floating-point and native DSP floating-point formats is made by DSGetMem or DSPutMem. This may change in later versions of Hardware Library, as new uMemType options are added. Thus, for example, if a calling MATLAB application is transferring 64-bit floating-point data to a DSP program, the DSP program may need to "know" to convert this data to its native 32-bit floating-point format before processing. Note, however, that for 32-bit floating-point DSP devices which do not use IEEE format (which is the default format used in Borland, Microsoft, and WATCOM C/C++ compilers), the DSDSPToIEEE and DSIEEEToDSP routines are available for converting native DSP 32-bit floating-point data to and from IEEE 32-bit floating-point format. Note also that these routines only convert data using calling application memory buffers, and do not involve DSP memory.

4) For 32-bit floating-point DSP devices which support byte- or word-addressable memory, addressing used by DSGetMem and DSPutMem appears as if DSP memory is accessible as contiguous locations each of uBitWidth bits. For example, if a DSP device supports byte-addressable memory, then using a uBitWidth value of DS_GM_SIZE16 would cause access to occur at the *byte address* of dwDSPDataAddr∗2, and a uBitWidth value of DS_GM_SIZE32 would cause access to occur at the byte address of dwDSPMemAddr∗4 (or the *word address* of dwDSPMemAddr∗2). Examples of byte-addressable

DSP devices include the AT&T DSP32C and DSP3210, and Texas Instruments TMS320C32.

5) The DS_GM_LINEAR_DATA_RT uMemType is intended for transferring 16-bit analog I/O data to DSP programs as fast as possible while the DSP device is in the run state. Currently, using the DS_GM_LINEAR_DATA_RT attribute will make a difference only with Texas Instruments TMS320 series devices. With all other devices, results are the same as using DS_GM_LINEAR_DATA. This attribute is provided because it turns out that data transfer for many TMS320 series based DSP/analog hardware designs is faster if only the high 16-bits of data memory are used when transferring data to DSP memory, or judicious use (or non-use) of the hold state built in to these devices is required. In all cases except for TMS320C25 devices, and situations where a Hypersignal DSP Source Code Interface operating mode is being used which expects high-word transfers, it is safest to use the DS_GM_LINEAR_DATA construct.

6) Currently, the DS_GM_VECTOR_DATA_X and DS_GM_VECTOR_DATA_Y uMemType attributes are only supported for Motorola DSP5600x and DSP9600x devices. Regardless of the DSP type, DS_GM_VECTOR_DATA_X will produce the same result as DS_GM_LINEAR_DATA.

7) Experience has shown that it would be difficult to specify a straightforward processor list parameter in DSP memory data transfer functions. For multiprocessor DSP/analog hardware designs, data memory is generally shared, and different processor data spaces are chosen using memory address values, which allows DSGetMem and DSPutMem to be used as is. In cases where data or program memory between processors is physically separate, DSWriteBoardReg and DSReadBoardReg calls may need to be made to select a memory space before using data transfer functions.

C/C++ Example

The source code example below transfers quantized sine wave (1000 kHz sine sampled at 100 kHz) data to DSP data memory address 0x800 from a calling application data buffer created by the Windows GlobalAlloc function. Note that the actual type of DSP memory receiving the data could be anything from 16-bit, 24-bit, or 32-bit data--the host buffer contains only quantized 16-bit values stored in 32-bit signed integers.

```
HGLOBAL hBuf;
long far* pBuf;
WORD wBuflen;
DWORD dwDSPBaseAddr;

dwDSPBaseAddr = 0x800;

hBuf = GlobalAlloc(GMEM_MOVEABLE, wBufLen*sizeof(long));
pBuf = (long far*)GlobalLock(hBuf);
for n = 0; n<wBufLen-1; n++) pBuf[n] = (long)32767∗sin(2∗pi∗n∗1000/10000);

UINT uStatus = DSPutMem(hBoard, DS_GM_LINEAR_DATA, dwDSPBaseAddr,
DSGetWordLength(hBoard), pBuf, wBuflen);

if (!uStatus) {

    HENGINE hEngine = DSGetBoardInfo(hBoard, DS_GBI_ENGINEHANDLE));
    int nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    itoa(nErrorType, tmpstr, 10);
    MessageBox(NULL, tmpstr, "DSPutMem error occurred", MB_OK);
}
GlobalUnlock(hBuf);  // release mem
```

Note in the above example that the DSGetWordLength function is used to transfer data using a data element width equal to the natural wordlength of the DSP device. This method is reliable mainly for integer data, and not floating-point data, because of the varying storage representations of floating-point (or fractional integer format) data used by popular DSPs.

See Also

DSGetMem

DSGetHVarMem

DSPutHVarMem

DSWaitForBuffer

WM_DSPENGINE_BUFRDY

DSGetWordLength

DSDSPToIEEE

DSIEEEToDSP

Related Users Guide Topics

section 4.3.4, DSP Memory Data Transfers

Example C/C++ Code: Processing Buffer Ready Messages, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSReadBoardReg

WORD DSReadBoardReg (hBoard,
wRegOffset)

```
HBOARD hBoard;          // DSP/analog hardware handle
WORD wRegOffset;        // offset from register base address
```

The DSReadBoardReg function reads a value from the specified register on the DSP/analog hardware indicated by hBoard.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
wRegOffset	offset from I/O base address of DSP/analog hardware

Returns

The return value is the current value of specified register on the DSP/analog hardware, assuming it can be read. Since register values can legitimately be NULL, the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should always be checked (see C/C++ source code example below).

Comments

In cases where directly accessing DSP/analog hardware is unavoidable, the DSReadBoardReg and DSWriteBoardReg functions may prove convenient.

In cases where registers are write-only, the return result is likely to be 0xffff. This may also be the case if a location is read which is actually not a board register. Consult the DSP/analog hardware documentation for detailed information about I/O address space registers the hardware may support.

C/C++ Example

```
wVal = DSReadBoardReg(hBoard, wOffset);

HENGINE hEngine = DSGetBoardInfo(hBoard, DS_GBI_ENGINEHANDLE);
int nErrorType = DSGetEngineErrorStatus(hEngine);
if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();

if (nErrorType != NULL) {

    // do error processing...
}
```

See Also

DSWriteBoardReg

DSSetBoardBaseAddr

DSAssignBoard

Related Users Guide Topics

I/O Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSResetBoard

UINT DSResetBoard (hBoard)

```
HBOARD hBoard;                  // DSP/analog hardware handle
```

The DSResetBoard function places all DSP processors on the DSP/analog hardware specified by hBoard into the reset state.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call

Returns

The return value is non-zero if all DSP processors were reset successfully. Otherwise, the return value is NULL, indicating the DSResetBoard call failed.

Comments

Upon successful return, all DSP devices on the DSP/analog hardware will be in the reset state.

All DSP devices are reset simultaneously, if the DSP/analog hardware supports a simultaneous-write mode. For multiprocessor hardware, if only certain DSP processors need to be reset, use the DSResetProcessor function.

The DSP Engine is responsible for accounting for necessary delays or special timing due to reset generators or other reset ciruitry.

For a few types of hardware, the DSResetBoard call may actually return with DSP devices in the run state, in which case a resident "host interface" DSP code segment will be active. This is necessary in cases where the design of the hardware and/or DSP device does not allow onboard memory to be accessed while the DSP device(s) are held in reset.

Note that in all cases, if the DSResetBoard function is successful, further operations such as DSGetMem and DSPutMem can be executed at any time. *If not, then further operations will produce undefined results*.

NOTE: DSResetBoard should not be called unless a successful DSInitBoard function has been completed previously.

C/C++ Example

See Also

DSResetProcessor

DSRunBoard

DSRunProcessor

DSInitBoard

Related Users Guide Topics

section 4.3.1, Hardware and Processor Control

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSResetProcessor

UINT DSResetProcessor (hBoard,
wProcessorList)

HBOARD hBoard; // DSP/analog hardware handle

WORD wProcessorList; // value specifying which processors should be reset

The DSResetProcessor function places one or more processors on the DSP/analog hardware specified by hBoard into the reset state. The wProcessorList parameter is used to indicate which processors should be reset.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
wProcessorList	value containing processor list; each bit represents a processor, starting with bit 0 for processor 0; a 1 causes the processor to be included, a 0 causes it to be unaffected

Returns

The return value is non-zero if the DSP processors specified by wProcessorList were reset successfully. Otherwise, the return value is NULL, indicating the DSResetProcessor call failed.

Comments

The DSResetProcessor function will return NULL in the following cases:

1. the wProcessorList parameter indicates more than processor 0, and the DSP/analog hardware has only one processor
2. the wProcessorList parameter is zero
3. the wProcessorList parameter has more bits set to one than there are processors on the DSP/analog hardware

Upon successful return, DSP devices on the DSP/analog hardware specified by wProcessorList will be in the reset state.

DSP devices specified by wProcessorList are reset simultaneously, if the DSP/analog hardware supports a simultaneous-write mode. For multiprocessor hardware, if all DSP processors need to be reset simultaneously, it is simpler to use the DSResetBoard function.

The DSP Engine is responsible for ensuring that necessary delays due to reset generators or other reset timing are accounted for.

For a few types of hardware, the DSResetProcessor call may actually return with DSP devices in the run state, in which case a resident "host interface" DSP code segment will be active. This is necessary in cases where the design of the hardware and/or DSP device does not allow onboard memory to be accessed while the DSP device(s) are held in reset.

Note that in all cases, if the DSResetProcessor function is successful, further operations such as DSGetMem and DSPutMem can be executed at any time. *If not, then further operations will produce undefined results.*

NOTE: DSResetProcessor should not be called unless a successful DSInitBoard function has been completed previously.

C/C++ Example

See Also

DSResetBoard

DSRunProcessor

DSRunBoard

DSInitBoard

Related Users Guide Topics

section 4.3.1, Hardware and Processor Control

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSRunBoard

UINT DSRunBoard (hBoard)

```
HBOARD hBoard;                  // DSP/analog hardware handle
```

The DSRunBoard function places all DSP processors on the DSP/analog hardware specified by hBoard into the run state.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call

Returns

The return value is non-zero if all DSP processors were started successfully. Otherwise, the return value is NULL, indicating the DSRunBoard call failed.

Comments

Upon successful return, all DSP devices on the DSP/analog hardware will be in the run state.

All DSP devices are started simultaneously, if the DSP/analog hardware supports a simultaneous-write mode. For multiprocessor hardware, if only certain DSP processors need to be started, use the DSRunProcessor function.

Note that in all cases, if the DSRunBoard function is successful, data transfer operations such as DSGetMem and DSPutMem may still be executed at any time. *If not, then further operations will produce undefined results*.

C/C++ Example

See Also

DSRunProcessor

DSResetBoard

DSResetProcessor

DSInitBoard

DSDisableBoard

Related Users Guide Topics

section 4.3.1, Hardware and Processor Control

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSRunProcessor

UINT DSRunProcessor (hBoard,
wProcessorList)

```
HBOARD hBoard;          // DSP/analog hardware handle
WORD wProcessorList;    // value specifying which processors should
                        be run
```

The DSRunProcessor function places one or more processors on the DSP/analog hardware specified by hBoard into the run state. The wProcessorList parameter is used to indicate which processors should be started.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
wProcessorList	value containing processor list; each bit represents a processor, starting with bit 0 for processor 0; a 1 causes the processor to be included, a 0 causes it to be unaffected

Returns

The return value is non-zero if the DSP processors specified by wProcessorList were started successfully. Otherwise, the return value is NULL, indicating the DSRunProcessor call failed.

Comments

The DSRunProcessor function will return NULL in the following cases:

1. the wProcessorList parameter indicates more than processor 0, and the DSP/analog hardware has only one processor

2. the wProcessorList parameter is zero

3. the wProcessorList parameter has more bits set to one than there are processors on the DSP/analog hardware

Upon successful return, DSP devices on the DSP/analog hardware specified by wProcessorList will be in the run state.

DSP devices specified by wProcessorList are started simultaneously, if the DSP/analog hardware supports a simultaneous-write mode. For multiprocessor hardware, if all DSP processors need to be started simultaneously, it is simpler to use the DSRunBoard function.

Note that in all cases, if the DSRunProcessor function is successful, data transfer operations such as DSGetMem and DSPutMem can be executed at any time. *If not, then further operations will produce undefined results*.

C/C++ Example

See Also

DSRunBoard

DSResetProcessor

DSResetBoard

DSInitBoard

DSDisableBoard

Related Users Guide Topics

section 4.3.1, Hardware and Processor Control

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSSetBoardBaseAddr

UINT DSSetBoardBaseAddr (hBoard,
uAddrType,
wBaseAddr)

HBOARD hBoard; // DSP/analog hardware handle

```
UINT  uAddrType;        // type of base address to set
WORD  wBaseAddr;        // base address value
```

The DSGetBoardBaseAddr function sets the current I/O or memory base address for the specified DSP/analog hardware.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
uAddrType	type of base address to set; can be one of the values defined below:

Value	Meaning
DS_GBBA_IOADDR	retrieve I/O base address
DS_GBBA_MEMADDR	retrieve memory base address

wBaseAddr	value to which to set base address

Returns

The return value TRUE if the I/O or memory base address was set successfully for the DSP/analog hardware specified by hBoard. Otherwise, the return value is NULL, indicating the DSSetBoardBaseAddr call failed.

Comments

DSSetBoardBaseAddr can be used to set base addresses for DSP/analog hardware at any time. Note these values were initialized in the original DSAssignBoard call which created hBoard. Note also that if either the uIOBaseAddr or uMemBaseAddr parameters in the original DSAssignBoard calls were NULL, then these values are initialized to the current I/O Base Address and Memory Base Address values in the Hardware Setup File.

The DSGetBoardBaseAddr call can be used to retrieve DSP/analog hardware base address values at any time. The values returned are either those initialized in the original DSAssignBoard call or set subsequently using a DSSetBoardBaseAddr call.

C/C++ Example

See Also

DSGetBoardBaseAddr

DSAssignBoard

Related Users Guide Topics

I/O Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Memory Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSSetBoardBusType

UINT DSSetBoardBusType (hBoard,
uBusType)

```
HBOARD hBoard;      // DSP/analog hardware handle
UINT uBusType;      // type of base address to set
```

The DSSetBoardBusType function sets the current bus type for the specified DSP/analog hardware.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
uBusType	bus type to set; can be one of the values defined below:

Value	Meaning
DS_AB_PCXT	PC-XT bus type
DS_AB_PCAT	PC-AT bus type
DS_AB_ENABLE80X86INST	enable 80x86 instruction set operations

Returns

The return value TRUE if the bus type was set successfully for the DSP/analog hardware specified by hBoard. Otherwise, the return value is NULL, indicating the DSSetBoardBusType call failed.

Comments

Depending on the speed of the host PC, and the type of DSP/analog hardware, there may be cases where specifying a different bus type can make the difference between correct operation of the hardware, or improve performance. For example, there are some DSP/analog hardware types which do not operate reliably when 80x86 type "string move" instructions are used, in which case the DS_AB_ENABLE80X86INST bus type attribute shown above should be avoided. For complete bus type information for specific hardware, see chapter 3 in the Hypersignal-Macro Series Hardware Reference Guide.

DSSetBoardBusType can be used to set bus types for DSP/analog hardware at any time. Note this value is initialized in the original DSAssignBoard call which created hBoard. Note also that if either the uBusType parameter in the original DSAssignBoard calls was NULL, then this value is initialized to the current Bus Type value in the Hardware Setup File.

The DSGetBoardBusType call can be used to retrieve DSP/analog hardware bus type values at any time. The value returned is either that initialized in the original DSAssignBoard call or set subsequently using a DSSetBoardBusType call.

C/C++ Example

See Also

DSGetBoardBusType

DSAssignBoard

Related Users Guide Topics

Bus Type Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSWaitForBuffer

UINT DSWaitForBuffer (hBoard,
nBufferNum,
dwFlagAddr,
uNotifyType)

```
HBOARD  hBoard;          // DSP/analog hardware handle
short int  nBufferNum;   // buffer number
DWORD  dwFlagAddr;       // flag address in DSP memory
UINT  uNotifyType;       // notification type
```

The DSWaitForBuffer function establishes a DSP Engine wait-for-buffer request with the specified DSP/analog hardware.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
nBufferNum	number of buffer to wait for
dwFlagAddr	for user-defined buffers only, address of buffer flag in DSP memory
uNotifyType	notification type; can be one of the values defined below:

Value	Meaning
DS_WFB_POLLED	wait request is polled mode
DS_WFB_INT	wait request is interrupt-driven

Returns

The return value TRUE if the wait-for-buffer request was established successfully. Otherwise, the return value is NULL, indicating the DSWaitForBuffer call failed.

Comments

A wait-for-buffer request is established with a DSP Engine by calling DSWaitForBuffer. Below is a short source code example (see also C/C++ Example below):

```
DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETDSPDATARDYMSG, hwnd);
DSWaitForBuffer(hBoard, nBufferNum, dwFlagAddr, uNotifyType);
```

In the above calls, hwnd is first registered to receive buffer ready and semaphore ready messages from the DSP Engine; in this case, hwnd will receive WM_DSPENGINE_BUFRDY messages.

nBufferNum values 0 through 15 are reserved for standard DSP Engine high-level function "data nodes", which are pre-defined sources and sinks of analog input and output data. In these cases, dwFlagAddr and uNotifyType are ignored. For example, nBufferNum values of 0 and 1 indicate continuous analog input data, alternating between buffers 0 and 1. uNotifyType is ignored for reserved nBufferNum values because DSP Engines may use specialized methods of buffer-checking, which may involve hardware register bits or other optimized techniques (not necessarily DSP board memory). The concept behind reserved data nodes is that more than one calling application may register a window handle to receive the same buffer ready messages.

Values for nBufferNum not equal to 0 through 15 can be used to set up user-defined buffer ready messages. In these cases, dwFlagAddr and uNotifyType must be specified.

If a uNotifyType value of DS_WFB_POLLED is specified, the DSP Engine will periodically check to see if the specified buffer is ready, and cause WM_DSPENGINE_BUFRDY messages to be sent to hwnd as needed. If a uNotifyType value of DS_WFB_INT is specified, the DSP Engine will set up and enable a host PC interrupt, and send a WM_DSPENGINE_BUFRDY message when the interrupt occurs.

For more detailed information, see Buffer Ready Messages from DSP Engines, in Signalogic DSP Software Users Guide section 7.4, Registering Callback Messages from DSP Engines.

C/C++ Example

Below is another source code example taken from dscope.cpp, showing partial code for creating a window and designating it to receive buffer ready messages:

```
// create window that can be used for callback messages
hwnd = CreateWindow(wndclass.lpszClassName,
        "Example DSPower-HWLib Test Program",
        WS_OVERLAPPEDWINDOW | WS_VSCROLL | WS_HSCROLL |
        WS_CLIPCHILDREN | WS_VISIBLE,
        rectScopePos.left,
        rectScopePos.top,
        nWidth,
        nHeight,
        NULL,
        NULL, // no menu
        hInstance,
        NULL);

ShowWindow(hwnd, nCmdShow);
UpdateWindow(hwnd);

// scope window is born; turn the board loose...
DSRunProcessor(hBoard, 0x01);  // processor 0 only

// register window to receive buffer ready messages, tell engine to wait for input data

DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETDSPDATARDYMSG, hwnd);
DSWaitForBuffer(hBoard, 0, NULL, DS_WFB_POLLED);

// no choice now but to follow Alice...
while (GetMessage(&msg, NULL, 0, 0)) {

    TranslateMessage (&msg);
    DispatchMessage (&msg);
};
```

See Also

WM_DSPENGINE_BUFRDY

DSCancelWaitBuffer

DSWaitForFlag

DSGetMem

DSPutMem

Related Users Guide Topics

Buffer Ready Messages from DSP Engines, in section 7.4 Registering Callback Messages from DSP Engines

Example C/C++ Code: Processing Buffer Ready Messages, in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software

DSWaitForFlag

UINT DSWaitForFlag (hBoard,
nFlagNum,
dwFlagAddr,
uNotifyType)

```
HBOARD  hBoard;          // DSP/analog hardware handle
short int  nFlagNum;     // flag number
DWORD  dwFlagAddr;       // flag address in DSP memory
UINT  uNotifyType;       // notification type
```

The DSWaitForFlag function establishes a DSP Engine wait-for-semaphore request with the specified DSP/analog hardware.

See below for function parameter descriptions.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
nFlagNum	number of flag to wait for
dwFlagAddr	for user-defined semaphores only, address of flag in DSP memory
uNotifyType	notification type; can be one of the values defined below:

Value	Meaning
DS_WFB_POLLED	wait request is polled mode
DS_WFB_INT	wait request is interrupt-driven

Returns

The return value TRUE if the wait-for-flag request was established successfully. Otherwise, the return value is NULL, indicating the DSWaitForFlag call failed.

Comments

A wait-for-semaphore request is established with a DSP Engine by calling DSWaitForFlag. Below is a short source code example (see also C/C++ Example below):

```
DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETDSPDATARDYMSG, hwnd);
DSWaitForFlag(hBoard, nFlagNum, dwFlagAddr, uNotifyType);
```

In the above calls, hwnd is first registered to receive buffer ready and semaphore ready messages from the DSP Engine; in this case, hwnd will receive WM_DSPENGINE_FLGRDY messages.

nFlagNum values 0 through 15 are reserved for standard DSP Engine high-level function "semaphore nodes", which are pre-defined status indicators for standard types of analog input and output data and acceleration processing. In these cases, dwFlagAddr and uNotifyType are ignored. For example, an nFlagNum value of 0 indicates accelerated file-based time domain data processing. uNotifyType is ignored for reserved nFlagNum values because DSP Engines may use specialized methods of flag-checking, which may involve hardware register bits or other optimized technique (not necessarily DSP board memory). The concept behind reserved semaphore nodes is that more than one calling application may register a window handle to receive the same semaphore messages.

Values for nFlagNum not equal to 0 through 15 can be used to set up user-defined semaphore messages. In these cases, dwFlagAddr and uNotifyType must be specified.

If a uNotifyType value of DS_WFB_POLLED is specified, the DSP Engine will periodically check to see if the specified semaphore is ready, and cause WM_DSPENGINE_FLGRDY messages to be sent to hwnd as needed. If a uNotifyType value of DS_WFB_INT is specified, the DSP Engine will set up and enable the next available DSP-to-host PC interrupt, and send a WM_DSPENGINE_FLGRDY message to hwnd when the interrupt occurs.

For more detailed information, see Semaphore Messages from DSP Engines, in Signalogic DSP Software Users Guide section 7.4, Registering Callback Messages from DSP Engines.

C/C++ Example

See Also

WM_DSPENGINE_FLGRDY

DSCancelWaitFlag

DSWaitForBuffer

DSGetMem

DSPutMem

Related Users Guide Topics

Semaphore Messages from DSP Engines, in section 7.4 Registering Callback Messages from DSP Engines

DSWriteBoardReg

WORD DSWriteBoardReg (hBoard,
wRegOffset,
wVal)

```
HBOARD hBoard;          // DSP/analog hardware handle
WORD wRegOffset;        // offset from register base address
WORD wVal;              // value to write
```

The DSWriteBoardReg function writes a value to the specified register on the DSP/analog hardware indicated by hBoard.

Parameter	Description
hBoard	DSP/analog hardware handle previously obtained from DSAssignBoard call
wRegOffset	offset from I/O base address of DSP/analog hardware
wVal	value to write to DSP/analog hardware register

Returns

The return value is TRUE if the value was written to the DSP/analog hardware register successfully. Otherwise, the value is NULL, indicating the DSWriteBoardReg function failed, and the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should be checked (see C/C++ source code example below).

Comments

In cases where directly accessing DSP/analog hardware is unavoidable, the DSReadBoardReg and DSWriteBoardReg functions may prove convenient.

C/C++ Example

```
if (!DSWriteBoardReg(hBoard, wRegOffset, 0x59a2)) {

    HENGINE hEngine = DSGetBoardInfo(hBoard, DS_GBI_ENGINEHANDLE);
    int nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();

    if (nErrorType != NULL) {

        // do error processing...
    }
}
```

See Also

DSReadBoardReg

DSSetBoardBaseAddr

DSAssignBoard

Related Users Guide Topics

I/O Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

2.2.4 Tool Library Functions

[@k functions, Tool Library, toollib.dll, tlib32.dll]

Below is a partial list of functions in Tool Library, which is a library of user-interface objects and controls used internally at Signalogic and by products such as DSPower-Block Diagram. Tool Library is also included with DSPower-HwLib examples and demos, to provide a minimal example user-interface across a wide variety of platforms, including various C/C++ programming environments (e.g. Borland, Microsoft, and WATCOM) and other programming environments such as Visual Basic . For information on how to include Tool Library DLLs (toollib.dll or tlib32.dll) in C/C++ and Visual Basic projects, see Signalogic DSP Software Users Guide section 8.1, Hardware Manager Access. The examples in this section refer to Hardware Manager, but

references to hwlib.h can be replaced by toollib.h or toollib.bas. In addition, some of the individual function descriptions below contain C/C++ source code examples showing how to use Tool Library.

Tool Library Functions

DSAddToolboxControl

DSCreateToolbox

DSDeleteToolbox

DSGetToolboxFlags

DSGetToolboxRect

DSGetToolLibErrorStatus

DSGetToolLibFlags

DSGetUserInput

DSInitToolLib

DSMarkApp

DSMessageBox

DSMoveToolbox

DSSendMessageControl

DSSendMessageToolboxControl

DSSetToolboxFlags

DSSetToolLibFlags

DSAddToolboxControl

HCONTROL DSAddToolboxControl (hToolbox,
ID,
ControlType,
pszLabel,
pszHelpText,
pCtrlItemsUser)

```
HTOOLBOX hToolbox;          // toolbox handle
WORD ID;                    // control ID
DWORD ControlType;          // control types
LPCSTR pszLabel;            // pointer to text string
                            containing label of the control
LPCSTR pszHelpText;         // pointer to text string
                            containing fly-overr help text
                            for the control
PCTRLITEMS pCtrlItemsUser;  // pointer to optional
                            CTRLITEMS structure
```

The DSAddToolboxControl function adds user-interface objects and control to a previously created toolbox. The toolbox may be thought of as a "container", which automatically sizes and shapes depending on the controls added. Objects and controls include toolbar buttons, pop-up dialogs, edit fields, knobs, dials, meters, list boxes, etc.

See below for DSAddToolboxControl calling parameter descriptions.

Parameter	Description
hToolbox	handle to toolbox previously obtained by DSCreateToolbox function
ID	user-specified control ID which can be used to identify control in WM_COMMAND messages or Visual Basic events; also pre-defined constants are available which specify toolbar button types and other
ControlType	type of specified control as defined in toollib.h and toollib.bas; see DS_ATC_xxx constants
pszLabel	pointer to text string which contains button text (for non bitmap-buttons), label for combo/edit/radio group, bitmap handle or bitmap filename for toolbar button, etc.
pszHelpText	pointer to null-terminated text string which contains fly-over help text, if any; default value is NULL
pCtrlItemsUser	optional pointer to CTRLITEMS structure which contains combo box or listbox items, pop-up dialog items, etc; default value is NULL

Returns

The return value is the handle to the new toolbox control, if successful. Otherwise, the return value is NULL, indicating that the control could not be created.

For further error checking, use the DSGetToolLibErrorStatus routine.

Comments

There are many pre-defined toolbox control and object types available in Tool Library. These are designated as DS_ATC_xxx, where "xxx" indicates the type of object.

For toolbar buttons and other image or pictoral (non-text) toolbox controls, the pszLabel parameter may be used to specify bitmap handles, if the high word of pszLabel is NULL, or bitmap filenames (.bmp, .pcx, or .tif) if the parameter is a string. Also, the ID parameter may be used to specify pre-defined bitmap types contained in Tool Library; see the DS_ATC_xxx values in toollib.h or toollib.bas, under the comment "built-in button IDs".

The DSAddToolboxControl function adds objects to a toolbox sequentially. The pre-defined DS_ATC_NEWLINE and DS_ATC_SEPARATOR control types can be used to control the spacing and "shape" the toolbox, depending on the toolbox type (box, horizontal bar, vertical bar, etc; see DSCreateToolbox function).

The toolbox size is automatically adjusted as new controls and objects are added. For example, adding one very large control will cause the toolbox to expand, as a rectangle, to accomodate that control in width and height, regardless of the size of other controls.

A call to the DSSetToolboxFlags function, with DS_GTF_TOOLBOXENABLE attribute set, is required to enable a toolbox and make it visible. Without this, no controls or user-objects added to the toolbox will be visible.

The DS_GTF_MENUENABLE attribute will cause a menu item to be added to the toolbox's owner window (unless the owner window is a child window) for each control or user-object. In this case, the label of the menu item will be the pszLabel parameter.

C/C++ Examples

The examples below show several DSAddToolboxControl function calls being used to add toolbar buttons, edit fields, and other simple controls and user objects to a toolbox. The toolbox handle is assumed

to be valid.

```
HWND hdlg;
HTOOLBOX    hPopupBox;
HTOOLBOX    hToolbox;
HCTRLITEMS  hCtrlItems;
PCTRLITEMS  pCtrlItems;

SendMessage(hdlg, WM_SETICON, FALSE,
  (LPARAM)LoadIcon((HINSTANCE)GetWindowLong(hdlg, GWL_HINSTANCE),
  "Smiley"));

// create a toolbar to use for some basic functions
hToolbox = DSCreateToolbox(hdlg, "Controls",
  DS_CT_HBAR | DS_CT_STRIPSTYLE, 0, 0);

// Create a simple toolbar button
DSAddToolboxControl(hToolbox, DS_ATC_VERTSCALEUP,
    DS_ATC_MOMENTARYBUTTON, NULL, NULL, NULL);
DSAddToolboxControl(hToolbox, NULL, DS_ATC_SEPARATOR, NULL,
  NULL, NULL);

// Create a popup box
hPopupBox = DSAddToolboxControl(hToolbox, DS_ATC_FSENTRY,
    DS_ATC_MOMENTARYBUTTON | DS_ATC_POPUP | DS_ATC_3DSTYLE,
    NULL, NULL, NULL);

// create and initialize a CTRLITEMS structure for use with popup control
hCtrlItems = GlobalAlloc(GMEM_ZEROINIT | GMEM_MOVEABLE,
  sizeof(CTRLITEMS));
pCtrlItems = (PCTRLITEMS)GlobalLock(hCtrlItems);

// create a label
DSAddToolboxControl(hPopupBox, NULL,
  DS_ATC_LABEL |DS_ATC_CENTERLABEL, "Num of channels", NULL,
    pCtrlItems);

// create a text box
lstrcpy(pCtrlItems->szCurEntry, "1");
DSAddToolboxControl(hPopupBox, NULL,
  DS_ATC_TEXTLEFTJUST | DS_ATC_EDITFIELD, NULL, NULL , pCtrlItems );

DSAddToolboxControl(hToolbox, NULL, DS_ATC_SEPARATOR, NULL, NULL,
  NULL);

DSSetToolboxFlags(hToolbox,
  DS_GTF_TOOLBOXENABLE | DS_GTF_MENUENABLE);
```

```
GlobalUnlock(hCtrlItems);
GlobalFree(hCtrlItems);
```

See Also

DSCreateToolbox

DSSendMessageControl

DSSendMessageToolboxControl

DSSetToolboxFlags

DSCreateToolbox

```
HTOOLBOX DSCreateToolbox     ( hWndOwner,
                               szTitle
                               dwType
                               x
                               y )

HWND hWndOwner;              // handle of owner window
LPCSTR szTitle;              // pointer to text string containing
                             title of the toolbox
DWORD dwType;                // type of toolbox
short int x;                 // x starting positon
short int y;                 // y starting position
```

The DSCreateToolBox function create a new toolbox. The toolbox may be thought of as a "container", which automatically sizes and shapes depending on the controls added. Objects and controls include toolbar buttons, pop-up dialogs, edit fields, knobs, dials, meters, list boxes, etc.

See below for DSCreateToolbox calling parameter descriptions.

Parameter	Description
hWndOwner	handle of owner window for the toolbox; thw owner window can be a main (overlapped or popup window) or child window
szTitle	text string for toolbox title; a NULL value specifies no

toolbox title

dwType — please refer to toollib.h

x — starting x position of toolbox, in owner window coordinates. Additional pre-defined values are shown below:

Value	Meaning
DS_CT_PREVBOX	use previous box for horizontal start position
DS_CT_PREVBOXADJACENT	use previous box for horizontal start position, adjacent
DS_CT_RIGHT	align toolbox with right of owner window

y — starting y position of toolbox, in owner window coordinates. Additional pre-defined values are shown below:

Value	Meaning
DS_CT_PREVBOX	use previous box for vertical start position
DS_CT_PREVBOXADJACENT	use previous box for vertical start position, adjacent
DS_CT_BOTTOM	align toolbox with bottom of owner window

Returns

The return value is handle to the toolbox created, if successful. Otherwise, the return value is NULL, indicating the owner handle is invalid.

For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSCreateToolBox function to create a toolbox:

```
HWND        hdlg;
HTOOLBOX hToolbox;

hToolbox = DSCreateToolbox(hdlg, "Controls",
  DS_CT_HBAR | DS_CT_STRIPSTYLE, 0, 0);
```

Note that

See Also

DSAddToolboxControl

DSDeleteToolbox

DSGetToolboxFlags

DSSetToolboxFlags

DSDeleteToolbox

UINT DSDeleteToolbox (hToolbox)

HTOOLBOX hToolbox // handle of toolbox

The DSDeleteToolbox function detete the toolbox with the handle hToolbox.

Parameter	Description
hToolbox	handle of toolbox to be destroyed

Returns

The return value is handle to the toolbox created, if successful. Otherwise, the return value is NULL, indicating the owner handle is invalid.

For further error checking, use the DSGetToolLibErrorStatus routine

(see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSDeleteToolbox function:

```
HWND        hdlg;
HTOOLBOX hToolbox;

hToolbox = DSCreateToolbox(hdlg, "Controls",
  DS_CT_HBAR | DS_CT_STRIPSTYLE, 0, 0);
if (hToolbox != NULL) DSDeleteToolbox(hToolbox);
```

Note that

See Also

DSCreateToolbox

DSGetToolboxFlags

UINT DSGetToolboxFlags (hToolbox)

HTOOLBOX hToolbox; // toolbox handle

The DSGetToolboxFlags function retrieves the flags of the toolbox.

Parameter	Description
Htoolbox	handle of toolbox

Returns

The return value is flags of the toolbox, if successful. Otherwise, the return value is NULL, indicating toolbox handle is invalid.

Value	Meaning
DS_GTF_TOOLBOXENABLE	// enable toolbox (show it)
DS_GTF_MENUENABLE	// enable toolbox's menu
DS_GTF_MOVEABLE	// make toolbox moveable/fixed
DS_GTF_TOPPARENTREL	// position and clip toolbox relative to top parent
DS_GTF_NONCLIENTAREA	// toolbox must be popup, because it will be located in non-client area
DS_GTF_MOVEWITHOWNER	// move if owner control is moved using DSMoveToolbox
DS_GTF_BORDER	// border attribute
DS_GTF_BORDERSPECIAL	// special red oval border used by DSPower-Block Diagram
DS_GTF_NOTVISIBLE	// toolbox should not be visible even after being enabled (becomes calling app's responsibility to control visibility)
DS_GTF_3DSTYLE	// activate 3D style (border, background)
DS_GTF_NOCLOSEBUTTON	// can be used to override Close button attribute
DS_GTF_NOOKBUTTON	// can be used to override OK button attribute
DS_GTF_SHADEDBKGND	// toolbox background is shaded (see DS_CT_SHADEDBKGND)
DS_GTF_NOAUTOSTICK	// disable "autostick", which occurs when toolbox shares two borders with owner window (toollib keeps it there, if owner window changes size)

For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSGetToolboxFlags function:

```
HWND hdlg;
HTOOLBOX hToolbox;
UINT uFlags;

hToolbox = DSCreateToolbox(hdlg, "Controls",
  DS_CT_HBAR | DS_CT_STRIPSTYLE, 0, 0);
uFlags = DSGetToolboxFlags(hToolbox);
```

Note

See Also

DSAddToolboxControl

DSCreateToolbox

DSSetToolboxFlags

DSGetToolboxRect

UINT DSGetToolboxRect (hToolbox, prect)

```
HTOOLBOX hToolbox;          // toolbox handle
RECT far* prect;            // pointer to rectangle coordinates
```

The DSGetToolboxRect function retrieve the client coordinate of the toolbox

Parameter	Description
Htoolbox	handle of toolbox
Prect	pointer to client rectangle coordinates

Returns

The return value is TRUE if successful. Otherwise, the return value is FALSE, indicating the toolbox handle is invalid.

For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSGetToolboxRect function to retrieve the client coordinate of toolbox rectangle:

```
HWND hdlg;
HTOOLBOX hToolbox;
RECT rect;

hToolbox = DSCreateToolbox(hdlg, "Controls",
  DS_CT_HBAR | DS_CT_STRIPSTYLE, 0, 0);

// return the client coordinate of toolbox
DSGetToolboxRect(hToolbox, &rect);
```

Note

See Also

DSAddToolboxControl

DSCreateToolbox

DSMoveToolbox

DSGetToolLibErrorStatus

UINT DSGetToolLibErrorStatus ( )

The DSGetToolLibErrorStatus function return the errors associated with running toollib.dll.

Parameter	Description
None	

Returns

The return value is errors contained in dllerr.h.

Value	Meaning
DSCOULDNOTCREATEBKGNDWINDOW	
DSCOULDNOTSTARTBKGNDTASK	
DSNOBKGNDWINDOW	
DSCOULDNOTINITIALIZEDLL	

Comments

The

C/C++ Example

Below is an example of using the DSGetToolLibErrorStatus function:

```
HWND hWnd;
UINT ret;

hWnd = DSInitToolLib();
```

if(hWnd == NULL) ret = DSGetToolLibErrorStatus();

Note that

DSGetToolLibFlags

UINT DSGetToolLibFlags ( )

The DSGetToolLibFlags function retrieves the flags that affects toollib.dll.

Parameter	Description
None	

Returns

The return value is flags associated with a toolbox.

Value	Meaning
DS_GTGF_SIZEINDEPENDENT	// object size independent of screen resolution
DS_GTGF_FLYOVERHELPENABLE	// enable fly-over help
DS_GTGF_FLYOVERHELPMINIMIZE	// enable short-form of fly-over help
DS_GTGF_ENABLECONTEXTHELP	// enable small context-sensitive help button
DS_GTGF_WIN31CONTEXTHELPLOOK	// enable Win 3.1x context-sensitive help look (solid yellow, instead of light yellow)

For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSGetToolLibFlags function:

```
HWND hWnd;
UINT uFlags;

hWnd = DSInitToolLib();
uFlags = DSGetToolLibFlags();
```

Note that

See Also

DSSetToolLibFlags

DSGetUserInput

int LIBAPI DSGetUserInput (hdlg,
szTitle
szLabel
szText
cbLen)

```
HWND hdlg;              // handle of owner dialog
LPCSTR szTitle;         // title of user input dialog
LPCSTR szLabel;         // label of edit field in user input
                        dialog
LPSTR szText;           // default text in edit field
int cbLen;              // length of edit field
```

The DSGetUserInput function initialize a dialog that retrieves information from the user.

Parameter	Description
Hdlg	handle of owner dialog
SzTitle	pointer to null-terminated text string which will contains title of the User Input dialog
SzLabel	pointer to null-terminated text string which will contains label of the Text Edit Field
SzText	pointer to null-terminated text string which will contains default text in the Text Edit field
CbLen	length of Text Edit field

Returns

The return value is the return value is the value of the nResult parameter given in the call to the EndDialog function used to terminate the dialog box, if successful.??? Otherwise, the return value is -1. For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSGetUserInput:

```
HWND hWnd;
```

```
char       szLabel[100], tmpstr1[256], tmpstr2[256];

// example code of using DSGetUserInput

lstrcpy(szLabel, "Please enter your name");
lstrcpy(tmpstr1, "John Smith");

DSGetUserInput(hdlg, "DSGetUserInput Example", szLabel, tmpstr1,
  sizeof(tmpstr1));

lstrcpy(szLabel, "DSGetUserInput");
lstrcpy(tmpstr2, "You name is ");
lstrcat(tmpstr2, tmpstr1);

DSMessageBox(NULL, tmpstr2, szLabel, MB_SYSTEMMODAL);
```

Note that

See Also

DSMessageBox

DSInitToolLib

HWND DSInitToolLib ( )

The DSInitToolLib function invokes toollib.dll and runs it in the background.

Parameter	Description
None	

Returns

The return value is handle to the toollib, if successful. Otherwise, the return value is NULL, indicating background handle is invalid. For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSInitToolLib:

```
DSInitToolLib();
DSSetToolLibFlags(DS_GTGF_FLYOVERHELPMINIMIZE |
  DS_GTGF_FLYOVERHELPENABLE);  // enable fly-over help
```

Note that

See Also

DSGetToolLibFlags

DSSetToolLibFlags

DSMarkApp

void DSMarkApp (hInst,
wVer)

HINSTANCE hInst;

WORD wVer;

The DSMarkApp function performs run time marking so that 16-bit version of software can use all the new functionality of their 32-bit counterpart.

Parameter	Description
Hinst	
WVer	

Returns

The return value is handle to the toolbox created if successful. Otherwise, the return value is NULL, indicating the Hardware Setup File index was out of range. For further error checking, use the

DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSMarkApp function:

```
BOOL fWin32;
HANDLE hInstance;

// set Win 3.1x / Win95 flag
fWin32 = ((LOWORD(GetVersion() & 0xff) == 3) &&
  ((LOWORD(GetVersion()) >> 8) >= 95));

// if Win95, mark app as 4.0
if (fWin32) DSMarkApp(hInstance, 0x0400);
```

Note that

See Also

DSGetToolLibFlags

DSInitToolLib

DSSetToolLibFlags

DSMessageBox

int DSMessageBox (hwndParent,
lpszText
lpszTitle
dwStyle)

HWND hwndParent;	// parent windle handle
LPCSTR lpszText;	// pointer to text string
LPCSTR lpszTitle;	// pointer to title string

DWORD dwStyle; // valid dialog style

The DSMessageBox function

Parameter	Description
hwndParent	Identifies the owner window of the message box to be created. If this parameter is NULL, the message box has no owner window.
lpszText	Points to a null-terminated string containing the message to be displayed.
lpszTitle	Points to a null-terminated string used for the dialog box title. If this parameter is NULL, the default title Error is used.
dwStyle	Specifies the contents and behavior of the dialog box. This parameter can be a combination of the following values:

Value	Meaning
MB_ABORTRETRYIGNORE	Message box contains Abort, Retry, and Ignore buttons
MB_APPLMODAL	User must respond to the message box before continuing work in the window identified by the hwndParent parameter
MB_APPLMODAL	Default value if neither MB_SYSTEMMODAL nor MB_TASKMODAL is specified
MB_DEFAULT_DESKTOP_ONLY	The desktop currently receiving input must be a default desktop; otherwise, the function fails. A default desktop is one an application runs on after the user has logged on
MB_DEFBUTTON1	First button is the default button. Note that the first button is always the default unless MB_DEFBUTTON2, MB_DEFBUTTON3, or MB_DEFBUTTON4 is specified
MB_DEFBUTTON2	Second button is a default button
MB_DEFBUTTON3	Third button is a default button
MB_DEFBUTTON4	The fourth button is a default button
MB_HELP	Windows 95 only: Adds a Help

	button to the message box. Choosing the Help button or pressing F1 generates a Help event
MB_ICONASTERISK	Same as MB_ICONINFORMATION
MB_ICONERROR	Windows 95 only: Same as MB_ICONHAND
MB_ICONEXCLAMATION	An exclamation-point icon appears in the message box
MB_ICONHAND	Same as MB_ICONSTOP
MB_ICONINFORMATION	An icon consisting of a lowercase letter i in a circle appears in the message box
MB_ICONQUESTION	A question-mark icon appears in the message box
MB_ICONSTOP	A stop-sign icon appears in the message box
MB_ICONWARNING	Windows 95 only: Same as MB_ICONEXCLAMATION
MB_OK	The message box contains one push button: OK
MB_OKCANCEL	The message box contains two push buttons: OK and Cancel.
MB_RETRYCANCEL	Message box contains two push buttons: Retry and Cancel.
MB_RIGHT	Windows 95 only : The text is right-justified
MB_RTLREADING	Windows 95 only : Displays message and caption text using right-to-left reading order on Hebrew and Arabic systems.
MB_SERVICE_NOTIFICATION	Windows NT only : The caller is a service notifying the user of an event. The function displays a message box on the current active desktop, even if there is no user logged on to the computer.If this flag is set, the hWnd parameter must be NULL. This is so the message box can appear on a desktop other than the desktop corresponding to the hWnd.
MB_SETFOREGROUND	Message box becomes the foreground window. Internally, Windows calls the SetForegroundWindow function for the message box.
MB_SYSTEMMODAL	All applications are suspended until the user responds to the

	message box. Unless the application specifies MB_ICONHAND, the message box does not become modal until after it is created; consequently, the owner window and other windows continue to receive messages resulting from its activation. Use system-modal message boxes to notify the user of serious, potentially damaging errors that require immediate attention (for example, running out of memory).
MB_TASKMODAL	Same as MB_APPLMODAL except that all the top-level windows belonging to the current task are disabled if the hWnd parameter is NULL. Use this flag when the calling application or library does not have a window handle available but still needs to prevent input to other windows in the current application without suspending other applications.
MB_YESNO	The message box contains two push buttons: Yes and No.
MB_YESNOCANCEL	The message box contains three push buttons: Yes, No, and Cancel.

Returns

The return value is zero if there is not enough memory to create the message box. If the function succeeds, the return value is one of the following menu-item values returned by the dialog box:

Value	Meaning
IDABORT	Abort button was selected
IDCANCEL	Cancel button was selected
IDIGNORE	Ignore button was selected
IDNO	No button was selected
IDOK	OK button was selected
IDRETRY	Retry button was selected
IDYES	Yes button was selected

If a message box has a Cancel button, the function returns the

IDCANCEL value if either the ESC key is pressed or the Cancel button is selected. If the message box has no Cancel button, pressing ESC has no effect.

The return value is handle to the toolbox created if successful. Otherwise, the return value is NULL, indicating the Hardware Setup File index was out of range. For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using DSMessageBox function

```
DSMessageBox(NULL, “Text in Message Box”, “Title text”,
  MB_OK | MB_ALWAYSONTOP|MB_SYSTEMMODAL);
```

Note that

See Also

DSGetUserInput

DSMoveToolbox

UINT DSMoveToolbox (hToolbox,
x
y
wd
ht)

```
HTOOLBOX hToolbox;      // handle of window
short int x;            // horizontal position
short int y;            // vertical position
```

```
short int wd;          // width
short int ht;          // height
```

The DSMoveToolbox function

Parameter	Description
hToolbox	Identifies the window.
x	Specifies the new position of the left side of the window.
y	Specifies the new position of the top of the window.
wd	Specifies the new width of the window, in pixels.
ht	Specifies the new height of the window, in pixels.

Returns

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. if toolbox handle is invalid.

The SetWindowPos function changes the size, position, and Z order of a child, pop-up, or top-level window. Child, pop-up, and top-level windows are ordered according to their appearance on the screen. The topmost window receives the highest rank and is the first window in the Z order.

SWP_NOACTIVATE Does not activate the window. If this flag is not set, the window is activated and moved to the top of either the topmost or non-topmost group (depending on the setting of the hwndInsertAfter parameter).

SWP_NOZORDER Retains the current Z order (ignores the hwndInsertAfter parameter).

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To get extended error information, call GetLastError.

Remarks

If the SWP_SHOWWINDOW or SWP_HIDEWINDOW flag is set, the window cannot be moved or sized.

All coordinates for child windows are client coordinates (relative to the upper-left corner of the parent window's client area).

A window can be made a topmost window either by setting the hwndInsertAfter parameter to HWND_TOPMOST and ensuring that the SWP_NOZORDER flag is not set, or by setting a window's position in the Z order so that it is above any existing topmost windows. When a non-topmost window is made topmost, its owned windows are also made topmost. Its owners, however, are not changed.

If neither the SWP_NOACTIVATE nor SWP_NOZORDER flag is specified (that is, when the application requests that a window be simultaneously activated and its position in the Z order changed), the value specified in hwndInsertAfter is used only in the following circumstances:

· Neither the HWND_TOPMOST nor HWND_NOTOPMOST flag is specified in hwndInsertAfter.

· The window identified by hwnd is not the active window.

An application cannot activate an inactive window without also bringing it to the top of the Z order. Applications can change an activated window's position in the Z order without restrictions, or it can activate a window and then move it to the top of the topmost or non-topmost windows.

If a topmost window is repositioned to the bottom

(HWND_BOTTOM) of the Z order or after any non-topmost window, it is no longer topmost. When a topmost window is made non-topmost, its owners and its owned windows are also made non-topmost windows.

A non-topmost window can own a topmost window, but the reverse cannot occur. Any window (for example, a dialog box) owned by a topmost window is itself made a topmost window, to ensure that all owned windows stay above their owner.

If an application is not in the foreground, and should be in the foreground, it must call the SetForegroundWindow function.

The return value is handle to the toolbox created if successful. Otherwise, the return value is NULL, indicating the Hardware Setup File index was out of range. For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSMoveToolbox:

```
HWND hdlg;
HTOOLBOX hToolbox;

hToolbox = DSCreateToolbox(hdlg, "Controls",
  DS_CT_HBAR | DS_CT_STRIPSTYLE, 0, 0);

// move toolbox left to 5, and top to 10 and retain original height and width
DSMoveToolbox(hToolbox, 5, 10, NULL, NULL);

// move toolbox left to 5, and top to 10 and change width to 100, height to 50
DSMoveToolbox(hToolbox, 5, 10, 100, 50);
```

Note that

See Also

DSAddToolboxControl

DSCreateToolbox

DSGetToolboxRect

DSSendMessageControl

LRESULT DSSendMessageControl (hControl,
uMsg
wParam
lParam)

HCONTROL hControl;
UINT uMsg;
WPARAM wParam;
LPARAM lParam; //

The DSSendMessageControl function

Parameter	Description
hControl	
uMsg	
wParam	
lParam	

Returns

The return value is handle to the toolbox created if successful. Otherwise, the return value is NULL, indicating the Hardware Setup File index was out of range. For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSSendMessageControl:

```
HWND hdlg;
HTOOLBOX hToolbox;
HTOOLBOX    hCheck1, hCheck2;

hToolbox = DSCreateToolbox(hdlg, "Controls",
  DS_CT_HBAR | DS_CT_STRIPSTYLE, 0, 0);

// Create simple checkboxes
hCheck1 = DSAddToolboxControl(hToolbox, NULL, DS_ATC_CHECKBOX ,
  "Unchecked" , "Check Box 1", NULL);
hCheck2 = DSAddToolboxControl(hToolbox, NULL, DS_ATC_CHECKBOX ,
  "Checked", "Check Box 2", NULL);

// Set check box state to unchecked
DSSendMessageControl(hCheck1, BM_SETCHECK, (WPARAM)0, 0L);
// Set check box state to checked
DSSendMessageControl(hCheck2, BM_SETCHECK, (WPARAM)1, 0L);
```

Note that

See Also

DSAddToolboxControl

DSSendMessageToolboxControl

DSSendMessageToolboxControl

LRESULT DSSendMessageToolboxControl (hToolbox,
ID
uMsg
wParam
lParam)

HTOOLBOX hToolbox;
WORD ID;
UINT uMsg;
WPARAM wParam;
LPARAM lParam; //

The DSSendMessageToolboxControl function

Parameter	Description
HToolbox	
ID	
UMsg	
WParam	
LParam	

Returns

The return value is handle to the toolbox created if successful. Otherwise, the return value is NULL, indicating the???. For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSSendMessageToolboxControl:

```
HWND hdlg;
HTOOLBOX hToolbox;
HCTRLITEMS  hCtrlItems;
PCTRLITEMS  pCtrlItems;

hToolbox = DSCreateToolbox(hdlg, "Controls",
  DS_CT_HBAR | DS_CT_STRIPSTYLE, 0, 0);
```

```
// create and initialize a CTRLITEMS structure for use with popup control
hCtrlItems = GlobalAlloc(GMEM_ZEROINIT | GMEM_MOVEABLE,
  sizeof(CTRLITEMS));
pCtrlItems = (PCTRLITEMS)GlobalLock(hCtrlItems);

// create a text box
// Set initial text to "11111111111"
lstrcpy(pCtrlItems->szCurEntry, "11111111111");
hText = DSAddToolboxControl(hToolbox, NULL,
  DS_ATC_TEXTLEFTJUST | DS_ATC_EDITFIELD, NULL, NULL , pCtrlItems );
DSSetToolboxFlags(hToolbox,
  DS_GTF_TOOLBOXENABLE | DS_GTF_MENUENABLE);

lstrcpy(pCtrlItems->szCurEntry, "Hello");  // Change control text to "Hello"
DSSendMessageToolboxControl(hToolbox, 100, WM_SETTEXT, 0L,
  (LPARAM)pCtrlItems->szCurEntry);

GlobalUnlock(hCtrlItems);
GlobalFree(hCtrlItems);
```

Note that

See Also

DSAddToolboxControl

DSSendMessageControl

DSSetToolboxFlags

UINT DSSetToolboxFlags (hToolbox,
Flags)

HTOOLBOX hToolbox;

UINT Flags;

The DSSetToolboxFlags function

Parameter	Description
Htoolbox	
Flags	

Returns

The return value is handle to the toolbox created if successful. Otherwise, the return value is NULL, indicating ?????

For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSSetToolboxFlags:

```
HWND hdlg;
HTOOLBOX hToolbox;

hToolbar = DSCreateToolbox(hdlg, "Controls",
  DS_CT_HBAR | DS_CT_STRIPSTYLE, 0, 0);
DSSetToolboxFlags(hToolbox,
  DS_GTF_TOOLBOXENABLE | DS_GTF_MENUENABLE);
```

Note that

See Also

DSCreateToolbox

DSGetToolboxFlags

DSSetToolLibFlags

void DSSetToolLibFlags (Flags)

UINT Flags;

The DSSetToolLibFlags function

Parameter	Description

Flags	#define DS_GTGF_SIZEINDEPENDENT 1
	// object size independent of screen resolution
	#define DS_GTGF_FLYOVERHELPENABLE 2
	// enable fly-over help
	#define DS_GTGF_FLYOVERHELPMINIMIZE 4
	// enable short-form of fly-over help
	#define DS_GTGF_ENABLECONTEXTHELP 8
	// enable small context-sensitive help button
	#define DS_GTGF_WIN31CONTEXTHELPLOOK
	16 // enable Win 3.1x context-sensitive help look
	(solid yellow, instead of light yellow)

Returns

The return value is handle to the toolbox created if successful. Otherwise, the return value is NULL, indicating ?????. For further error checking, use the DSGetToolLibErrorStatus routine (see C/C++ source code example below).

Comments

The

C/C++ Example

Below is an example of using the DSSetToolLibFlags:

```
DSInitToolLib();
DSSetToolLibFlags(DS_GTGF_FLYOVERHELPMINIMIZE |
  DS_GTGF_ENABLECONTEXTHELP);
```

Note that

See Also

DSGetToolLibFlags

DSInitToolLib

2.2.5 Signal Processing Library Functions

@k functions,, DSPower-SPLib, functions, Signal Processing Library, splib.dll]

This section not available yet.

2.3 DSP Source Code Functions

This section not yet available. For general information, please see Signalogic DSP Software Users Guide chapter 5, Using DSP Source Code Interface Software, which contains the general and specific topics sections of interest listed below:

General DSP Source Code Interface Software TopicsSections

5.0.1 DSP Source Code Interface Documentation

5.0.2 DSP Source Code Interfaces for Hypersignal

5.0.3 DSP Source Code Interfaces for Hypersignal in Engine Mode

Specific DSP Source Code Interface Software TopicsSections

5.7 Analog Devices ADSP-21xx Source Code Interface for Hypersignal

5.2 AT&T WE DSP32 Source Code Interface for Hypersignal

5.4 AT&T WE DSP32C Source Code Interface for Hypersignal

5.9 AT&T WE DSP3210 Source Code Interface for Hypersignal

5.3 Motorola DSP5600x Source Code Interface for Hypersignal

5.8 Motorola DSP96002 Source Code Interface for Hypersignal

5.6 NEC 77220 Source Code Interface for Hypersignal

5.1 Texas Instruments TMS320C25 Source Code Interface for Hypersignal

5.5 Texas Instruments TMS320C3x Source Code Interface for Hypersignal

5.10 Texas Instruments TMS320C4x Source Code Interface for Hypersignal

5.11 Texas Instruments TMS320C5x Source Code Interface for Hypersignal

Notes: